Dr. C. Wacko's

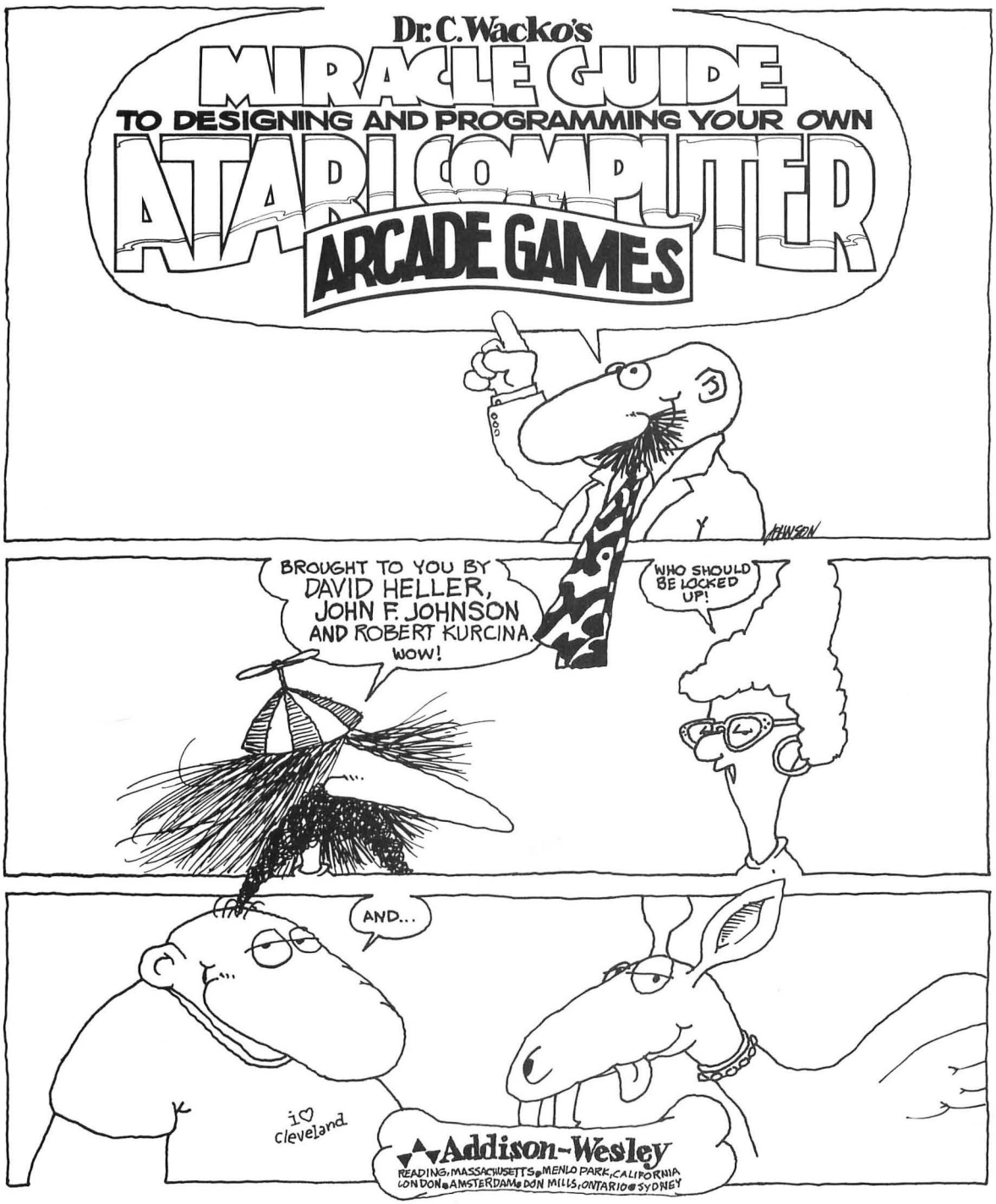
MIRACLE GUIDE

TO DESIGNING AND PROGRAMMING YOUR OWN

ATARI COMPUTER

ARCADE GAMES

IN CASE SOME IDIOT SPILLED CLAM DIP ON THE COVER, THIS IS...


Dr. C. Wacko's
MIRACLE GUIDE
TO DESIGNING AND PROGRAMMING YOUR OWN
ATARI COMPUTER
ARCADE GAMES
JOHNSON
BROUGHT TO YOU BY DAVID HELLER, JOHN F. JOHNSON AND ROBERT KURCINA. WOW!
WHO SHOULD BE LOCKED UP!
AND...
I ♡ Cleveland
Addison-Wesley
READING, MASSACHUSETTS • MENLO PARK, CALIFORNIA
LONDON • AMSTERDAM • DON MILLS, ONTARIO • SYDNEY

This book is in the
Addison-Wesley Microcomputer Books
Popular Series

Cover and Book Design-Teapot Graphics/John Johnson

Atari is a registered trademark of Atari, Inc.

Many thanks to Atari, Inc. for the art used in Appendix A, ATASCII Codes. Used with permission of Atari, Inc.

Library of Congress Cataloging in Publication Data

Heller, David L.
Dr. C. Wacko's miracle guide to designing and programming your own Atari computer arcade games.

Includes index.
1. Computer games. 2. Atari computer--Programming. 3. Basic (Computer program language) I. Johnson, John, 1944- . II. Kurcina, Robert. III. Title. IV. Title: Doctor C. Wacko's guide to designing and programming your own Atari computer arcade games.
GV1469.2.H43 1983 794.8'2 83-17257
ISBN 0-201-11488-7

Copyright © 1983 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

ISBN 0-201-11488-7
ISBN 0-201-11490-9 (book/disk package)

ABCDEFGHIJ-SE-876543
First printing, September 1983

Table of Contents

A Welcome Message from Dr. C. Wacko, Professor of Computer Wacko Science 1

1. Elementary, My Dear Wacko!: Game Design Elements 6

2. Graphics Modes, COLOR and PLOT Graphics, and Lots of Other Great Stuff 16

3. Character Graphics 46

4. Flip-Flop Animation 71

5. Movement 90

6. Taking Control with Your Joystick 99

7. The Big Frame-Up: Joystick-Controlled Animated Characters 108

8. Adversaries and Things that Bounce in the Night 119

9. Zounds 134

10. The Bogus Balonous Bonus Section: Player-Missile Graphics Made Simple 162

Appendix A: ATASCII Codes 189

Appendix B: Utility Programs 198

Appendix C: Myrtle the Turtle 216

Appendix D: Smokey Peek's Pokes & Peeks 228

Index 231

ACKNOWLEDGMENTS

This book wouldn't be a "book" if it weren't for three very special people:

Randy Biggs, for getting the project pointed in the right direction.

David Miller, editor par-excellance, for keeping it on track and seeing the project through with me.

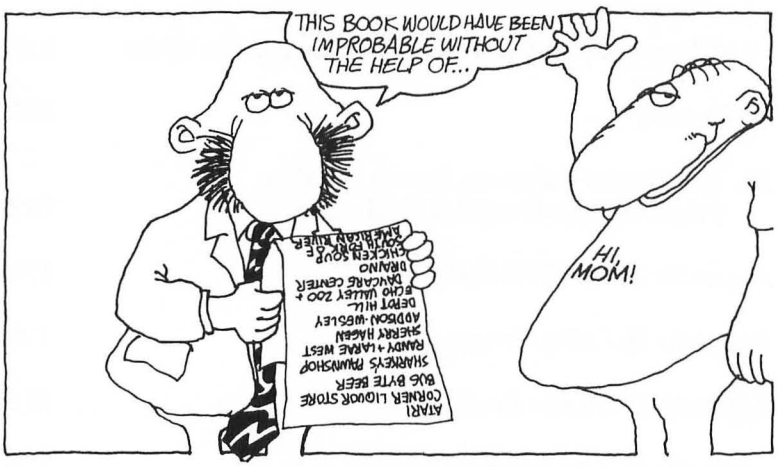
and

Dorothy Heller, for putting up with the real Dr. C. Wacko.

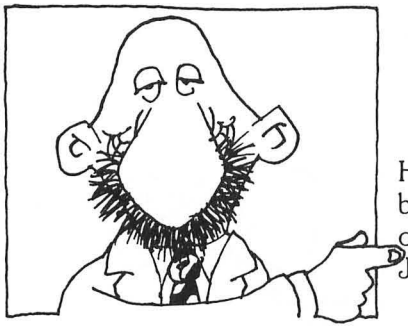
Mentioning Randy and LaRae West, Sherry Hagen, Gay Fairweather, and Draino won't hurt either.*

Thank you,

Dave Heller, John Johnson, & Robert Kurcina

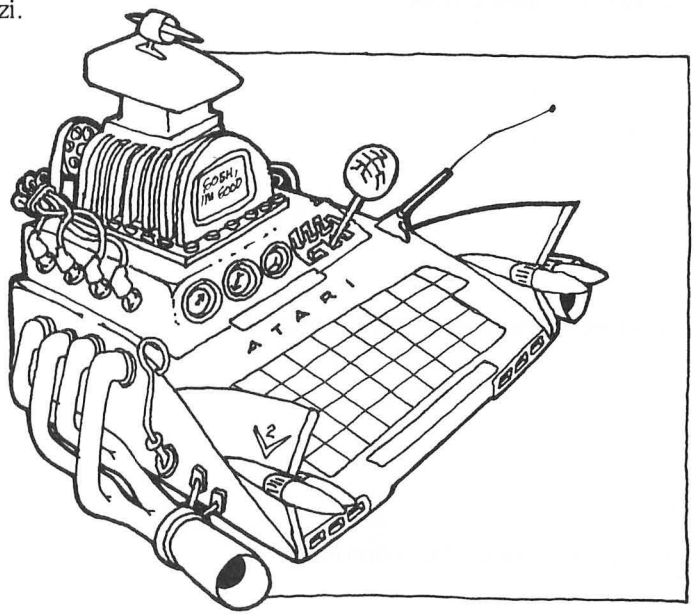


* HOWEVER, IT CERTAINLY WON'T HELP THEIR REPUTATIONS ANY.



A Welcome Message from Dr. C. Wacko Professor of Computer Wacko Science

Holy Zanzibar! Are you going to be glad that you bought this book! It will reveal *all*. All the inside tricks I've learned after years of research, tedious experimentation and conceptualizing in my Jacuzzi.



Your Atari computer is so talented, you'll soon be astounding your friends and neighbors with *your* own action-packed arcade games! (If you're a speedy reader.)

And here's the amazing part! Rocket ships will whiz across your brilliantly colored screen, chartreuse creatures of your own design will squirm obnoxiously under the control of your joystick, or the Atari Symphony will play in perfect harmony to your original composition—and it's all done in BASIC! The internationally famous Dr. C. Wacko shows you how. Trust me!

This highly edifying, educational, instructive, and informative book is filled with arcade game *action* elements: *ZAP! POW!* and *BRZZK!*

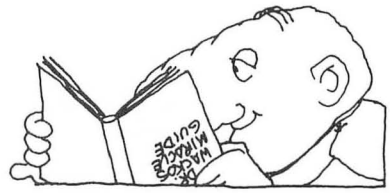
This book is filled with short BASIC routines, machine language subroutines, and entire games to show you how to design and program your own Atari arcade games.

This book is filled with *fun*!

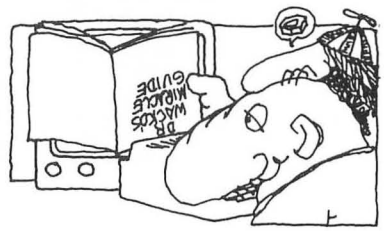
E-Z Book Operating Instructions

1. Know a little BASIC.

2. Open book.



3. Place book next to Atari computer.



4. Turn on your Atari computer.

5. Turn on your fingers.

6. Place your fingers on keyboard.

7. Enjoy your voyage through Dr. C. Wacko's action-packed universe of arcade game programming!

MURFG...
OY VEY
MY TURN TO PLAY SPACE DUCK!
?
SHERMAN TANK CLUB
OK, IDIOTS! BREAK IT UP— YOU'RE ACTING LIKE CHILDREN!
BESIDES, WE HAVE TO INTRODUCE OURSELVES...

DR. C. WACKO'S MIRACLE GUIDE

DR. WACKO

EVICTED FROM IOWA IN 1936, INVENTED FIRST WIND-UP COMPUTER IN 1947. INVENTED MIRACLE ARCADE SYSTEM BECAUSE PETUNIA WOULDN'T GIVE HIM ENOUGH QUARTERS, AND THINGS HAVE BEEN GOING BRZZK EVER SINCE.

MS. PEEKY

MIDWESTERN ROMANTIC LOOKING FOR A GOOD PEEK. SHE'LL INTRODUCE YOU TO SOME OF THE STRANGEST PEEKS IN TOWN. SHE WOULD RATHER BE IN A GOTHIC ROMANCE NOVEL.

MRS. PETUNIA WACKO

WACKO'S CHARMING AND ATTRACTIVE WIFE. AN ADVANCED MAINFRAME COMPUTER PROGRAMMER AND DR. WACKO'S MOST VOCAL CRITIC. SHE IS ALSO THE ONLY ONE IN THE GROUP WITH ANY PRACTICAL KNOWLEDGE.

SNIDLEY SEERSUCKER

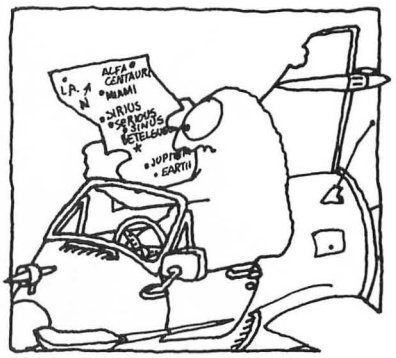
SPECIALIZES IN DASTARDLY PLOTS. HE WILL GUIDE YOU THROUGH THE WORLD OF PLOT GRAPHICS IN HIS OWN MELODRAMATIC WAY.

JUNIOR

LOVINGLY CALLED CEMENTHEAD BY HIS MANY FRIENDS, HES BEING GROOMED TO TAKE OVER THE BUSINESS WHEN HIS FOLKS RETIRE TO CLEVELAND. HE LOVES WATCHING TEST PATTERNS.

GROVER

AN ILLEGAL ALIEN WHO GOT LOST ON THE WAY TO ALPHA CENTAURI BECAUSE OF BENT WARP DRIVE. HIS '59 CADDY SPACE SHIP CAN'T FLY OVER 200 FEET BECAUSE CAPTAIN ACTION FIXED IT.

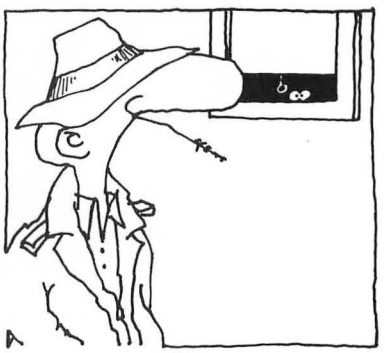


CAPTAIN ACTION

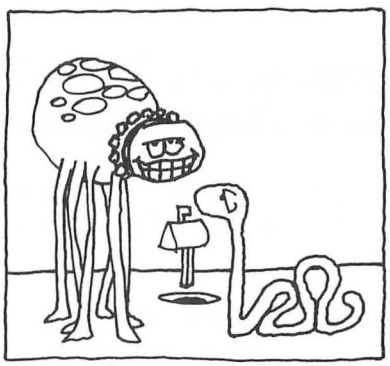
RESIDENT FIXIT & SWASHBUCKLING HERO OF GAME FREAKS, HE LEFT REALITY YEARS AGO TO WORK WITH DR. WACKO AND CAN'T FIND HIS WAY BACK.

SMOKEY PEEK

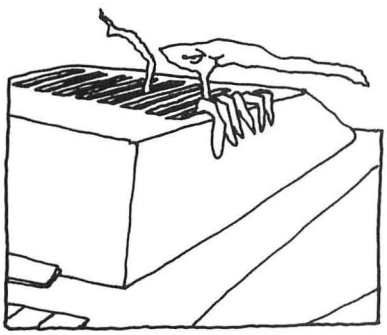
A RUGGED INDIVIDUALIST WHO'LL HAVE NOTHING TO DO WITH MS. PEEKY. HUMPHREY BOGART IS HIS HERO.



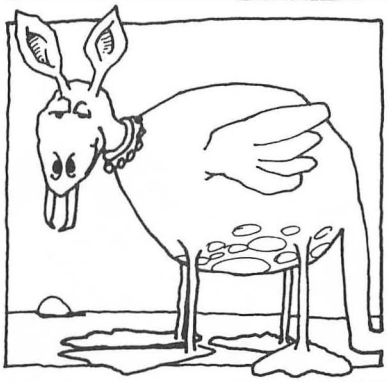
SLOW POKE
RIVALS JUNIOR FOR NATIVE INTELLIGENCE, BUT HAS BECOME A PEEK EXPERT AFTER THIRTEEN YEARS OF CORRESPONDENCE SCHOOL.



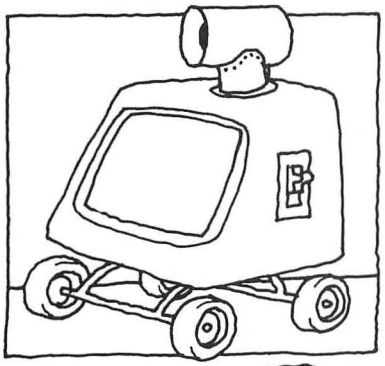
CUCURACHA & NIBBLES THE WORM
NIBBLES WILL SHOW YOU THE WORLD OF BITS AND BYTES. CUCURACHA CAN CAUSE BIG PROBLEMS IF YOU'RE NOT CAREFUL. A LITTLE BUG CAN MAKE A BIG MESS.



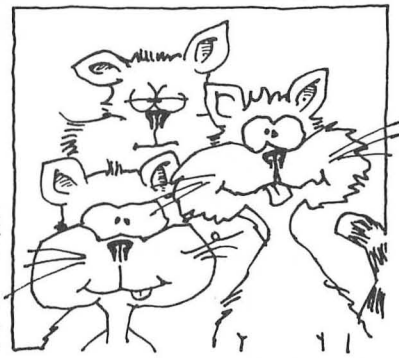
SUBTERRANEAN SUBROUTINE MAKER
LIVING DEEP WITHIN YOUR PROGRAM, THIS DENIZEN OF THE DISK WILL SHOW YOU SOME TRULY AMAZING SUBROUTINES. IF YOU'RE NICE.



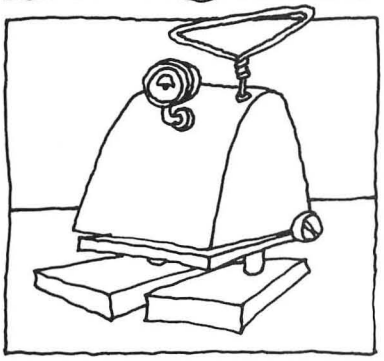
FIDO
CAME DOWN WITH GROVER BUT FEELS MORE AT HOME WITH JUNIOR, WHO'S HIS INTELLECTUAL EQUAL. HE'S LOYAL ONLY TO ANCHOVY BURRITOS.



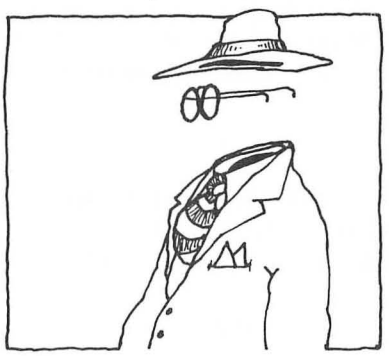
MACHINE LANGUAGE MACHINE
A MONSTROUS MARVEL OF THE MACHINE AGE, HE MUNCHES NUMBERS AND SPITS THEM OUT INTO SUPER FAST MACHINE LANGUAGE ROUTINES. USUALLY.



THE WACKO CATS
KEYS, PADDLES & JOYSTICK LOVE TO WALK ON THE COMPUTER AND MUNCH CONTROLS WHEN NOBODY IS LOOKING. TOTALLY UNHOUSEBROKEN.



CLARENCE COMPACTOR
SQUASHES LONG PROGRAMS INTO MANAGABLE SIZE. BUILT BY WACKO OUT OF A 1949 NASH AIRFLYTE.



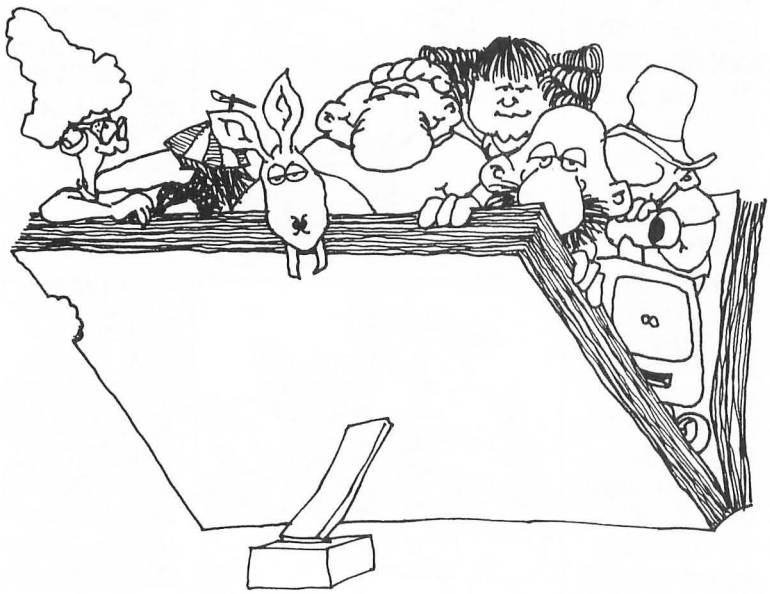
THE INVISIBLE MAN
TENDS TO SHOW UP IN UNEXPECTED PLACES AT SOME VERY ODD MOMENTS.

1
Elementary, My Dear Wacko: Game Design Elements

I'll bet you have a great imagination. Every time you play an arcade game, you probably think of ways to make it better and more fun to play.

I'm fortunate. I have a whole gang of weird cartoon characters living at my house who help me turn my imaginative ideas into great arcade games.

Now this strange gang is here between the pages of this book!



These weirdos can't replace your imagination, but they can give you lots of encouragement, concepts, and the tools you'll need to become a truly *creative* arcade game designer.

With their help, I'll show you:

- How to draw large images on the screen; including vibrantly colored playing fields
- How to build complex and colorful images called characters
- How to move these characters

- How to control their movement
- How to combine animation with movement
- How to generate characters that are completely out of the player's control
- How to use the special capabilities of your Atari—especially Player-Missile graphics—to make professional-looking arcade games
- How to work sound into your games to really top things off

Developing Themes

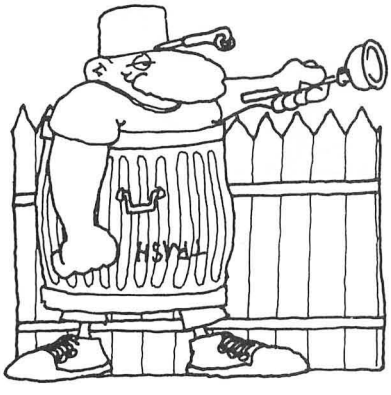
Programming is an *art* as well as a science. Writing a game program is almost like writing a novel. All the basic story elements must be present in your game to make it successful, believable, and fun to play, again, and again, and. . . .

After you've outlined your "game story," it's just a simple matter of selecting the right programming techniques to make it come alive. (Simple once you've read this book, that is.)

The first step in designing a winning computer game is to conceptualize a *theme*. The game theme is a short scenario that challenges the main character(s) to overcome *obstacles* to reach a definite *goal*. But to qualify as a true arcade game, your scenario must be loaded with *action*!

Junior does a great job of inventing action-packed game themes. He barged into my study last week and yelled, "Wow, dad! What do you think of these themes?":

1. Winning a duel against a Jedi-Knight
2. Abandoning ship and saving the children
3. Finding your way out of a complex maze
4. Rescuing your mate from Godzilla
5. Overcoming adversity to get to school on time



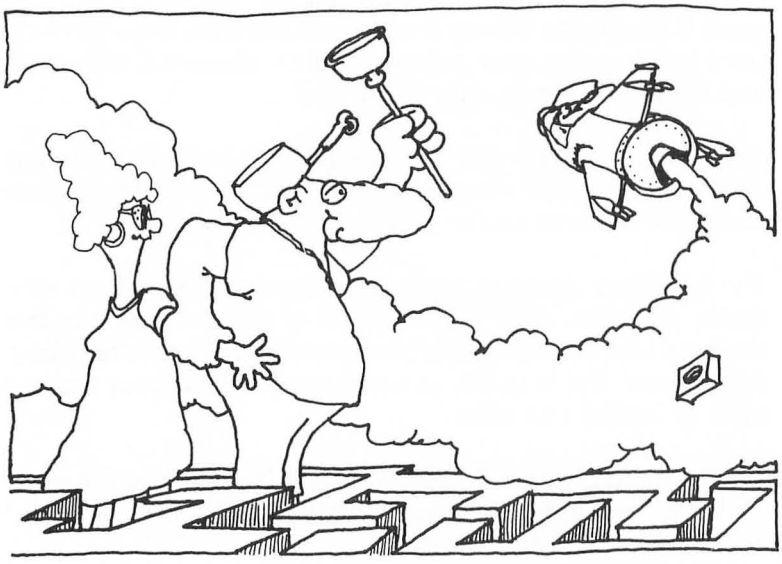
I thought they were great! They all contain action elements and most important, a definite goal. And each of Junior's themes can be graphically displayed on the computer's screen. Let's look at one of Junior's brilliant themes in more detail.

Junior's futuristic duel against a Jedi-Knight is loaded with great possibilities. The playing field could be the stylized interior of a spaceship or a rugged moonscape; just use your imagination!

This game could be a two-player game, or you might want to include an option that lets the player duel against the computer. Of course, you'll want to control your Jedi-Knights with a joystick.

Combining Themes

You can combine a number of themes to make your game more exciting and challenging. I decided to combine some of Junior's ideas by inventing a game called Dueling Hobos. In Dueling Hobos, the hero must survive a duel against an attacking Jedi-Knight as he searches for a way out of a complex maze after rescuing his girlfriend from Godzilla.



Another game variation using Junior's theme elements might be: "The ship is sinking. Your hero is lost in a maze of staterooms and must find his way to the deck. But his every move is blocked by a crazed Jedi-Knight." (Where did he come from again?)

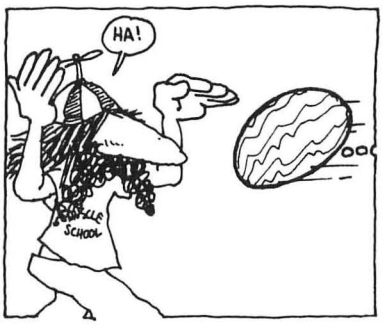
Draw From Your Own Experience

Drawing from your experiences is one of the best ways to conceptualize game themes. Quite often, games that are adapted from real-life situations are very successful because other people have shared the same experiences and can easily relate to your game. For example, a simple game molded around the theme of a student struggling to get to school on time (or get out of school early!) could be a smashing success.

Sports Themes, Etc.

Sports and the arts always make popular computer game themes. This type of game can be a simulation of the real thing (baseball, soccer, ballet, mountain climbing) or you can let your imagination run amuck and create situations in which your character tries to achieve a bizarre goal off the playing field.

Wacko Runs Amuck!



My imagination often runs amuck, and during one of my muckier days I created two game themes that Captain Action said are destined to become gold medal winners:

- A berserk karate instructor attempts to chop watermelons as they menacingly roll toward him from different parts of the screen.
- A slim ballerina twirls as fast as possible, sinking lower into the ground with each turn, in an attempt to strike oil.

Don't Be Afraid to Experiment

Experiment with many ideas to arrive at the theme you finally develop into your full-fledged game. Don't be afraid to follow my example and go wacko. Computer games can have any type of theme: surrealistic, realistic, futuristic, the past. But the game's theme must be challenging and believable—something that will hold the interest of any game player.

Analyze Your Favorite Arcade Games

Think about some of your favorite arcade games. What's the theme? What's the goal? Pac-Man™ is a good example. Pac-Man's theme requires that the little muncher eat up all the little dots around the screen while avoiding wandering ghosts—*obstacles*. In this game the action never stops as Pac-Man fights to survive. His *goal* is to move on to the next highest level of play and accumulate the highest score possible.

Salmon™ is an example of an arcade game that uses a real-life situation. Salmon's theme requires that a salmon fight its way upstream while avoiding bears, predatory birds, and fishermen. The salmon's goal is to mate at the end of its arduous journey. Just remember: A good theme, like a good story, requires that

your main character overcome obstacles to reach a definite goal. As an arcade game, it must be loaded with action.

Thematically Thpeaking

To help Junior graduate from themophyte to themophyle, (one who createth great themeth), I had to pull the plug on his computer and bribe him to sit at the kitchen table, pick up a pencil ("What's that, Dad?"), and put his thoughts down on paper. But as usual, Junior pleasantly suprised me by creating a list of simple, combined, and dowright complex themes.

If you can drag yourself away from your computer for a few minutes, you can have fun by developing themes around your imaginary characters. Let's work through one of Junior's to give you the idea then; after that, you're on your own.

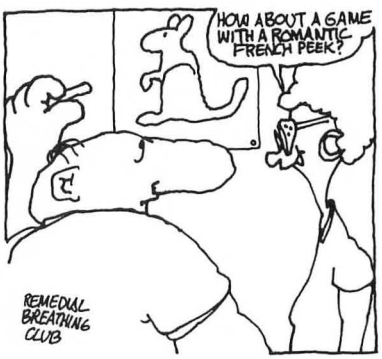
Junior's Theme Example

The Character: A Blue Kangaroo

The Theme: A hopping Blue Kangaroo tries to catch colored circles that appear at random positions on the screen. Each circle recharges his "life battery."

The Goal: Survival. The more circles he touches, the longer he lives and the higher the score.

The Obstacle: A ravenous DingoDog races around the screen trying to "eat" each colored circle before the Blue Kangaroo can reach it. The DingoDog becomes more ferocious as the game moves to higher levels. During the final levels he starts chasing the Blue Kangaroo—lots of action!



Experiment on your own until you have a good feel for theme development. Then move on to the next step: developing your characters.

Character Development

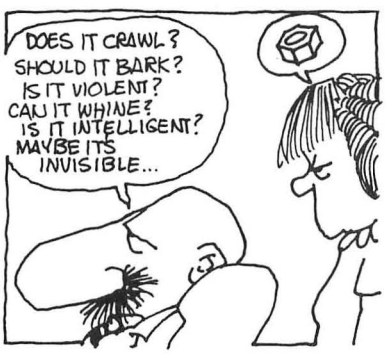
Now that you've developed your game theme and have a general idea what type of characters will act in your scenario, it's time to don your thinking cap and define each character in detail. In other words, give each of your characters some character!

Movin' and Groovin'

Take the time now to consider how your characters will move through your game scenario.

If you took one of Junior's suggestions and decided to design a game in which your main character must find his way out of a complex maze, he's certainly got to come equipped with movable legs. If your hero must duel against a Jedi-Knight, then he's got to have movable legs, flexible arms, and a sword. All this is fairly elementary, but, if you forge ahead without putting some thought into the basics, you may end up with a game similar to the disaster that Junior developed last week: A Pac-Man look-alike without a mouth!

Talk to Yourself



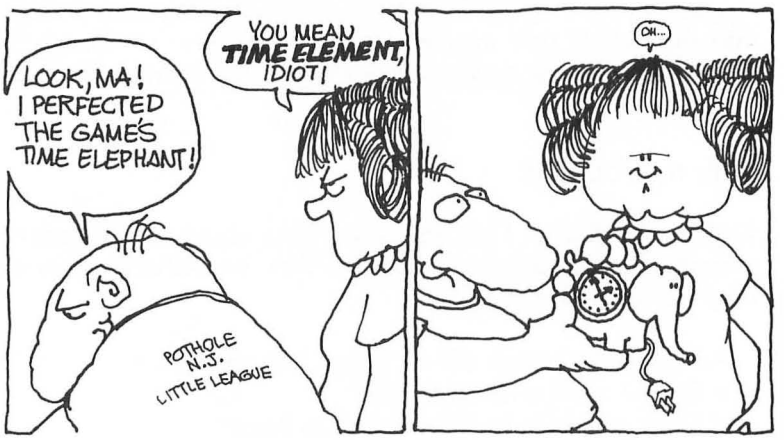
I talk to myself a lot. I ask myself questions about a character's attributes and write down my thoughts. Mrs. Wacko often wonders why I'm saying things like:

- How will it move? (Walk, jump, hop, skip etc.)
- Does it need a mouth? Eyes?
- Does it have to be able to turn its head?
- Does it have to fire a weapon?
- Use a sword? A gun and sword?
- Must it be able to pick up objects?
- Boy, am I hungry.

The Elephant of Time, Or, Time is Running Out, Or Is It?

There never seems to be enough time in my day to get everything done—so, when I design arcade games I vent my frustrations on the poor unsuspecting game player. Why should I be the only one who is limited by time?

Actually, the elephant of time. . . . Ooophs! Ahem, the *element of time* is an integral part of many popular arcade games.

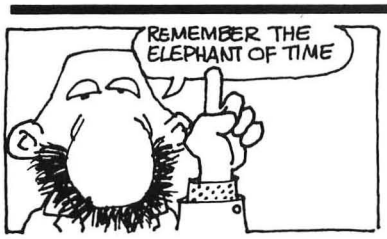


Visible Time: Readouts

In some popular arcade games, the passage of time is visually displayed by either a receding colored bar or a numerical readout that counts down toward zero. This "clock" forces the player to perform a task in an ever-decreasing amount of time. Atari's Lunar Lander™ for example, displays a numerical readout of fuel consumption. When fuel equals zero, time has run out and the game is over. Air Strike™ also uses fuel as a measure of time but, in this game it is displayed as a receding colored bar at the top of the screen. In Frogger™, the frog must arrive safely at its "breeding pond" within a specified amount of time, again displayed by a receding colored bar.

Invisible Time: No Readout

When Ms. Pac-Man™ munches a Power Pill she is awarded the *temporary* power to gobble the gobblins. After a short period of time her superpowers fade and Ms. Pac-Man must return to the defensive. When the hard-hat worker in Donkey Kong™ picks up



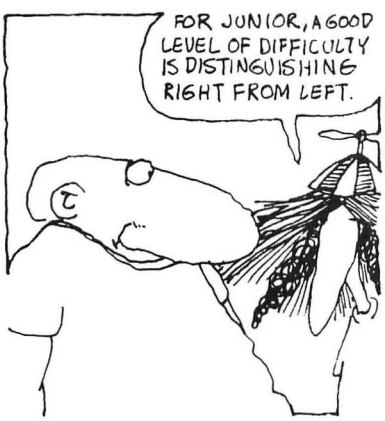
a Magic Mallet he is given the *temporary* power to crush the onrushing barrels. This superpower also fades after a short period of time.

Life Gets Harder Every Day

I've found that including *levels of increasing difficulty* in all my games serves two purposes:

- It lets the player begin at an easy level and become familiar with my game without becoming hopelessly frustrated.
- It offers an increasing challenge that will hold the player's interest and motivate him or her to master the game's higher levels.

In Chop Lifter™, more and deadlier adversaries are introduced into the action after each reconnaissance mission is completed. In Salmon, added obstacles are introduced to block the salmon's way after each successful upstream migration.



Other games increase in difficulty after the main character gains a certain number of points. But no matter what mechanism you use to increase your game's level of difficulty. . .

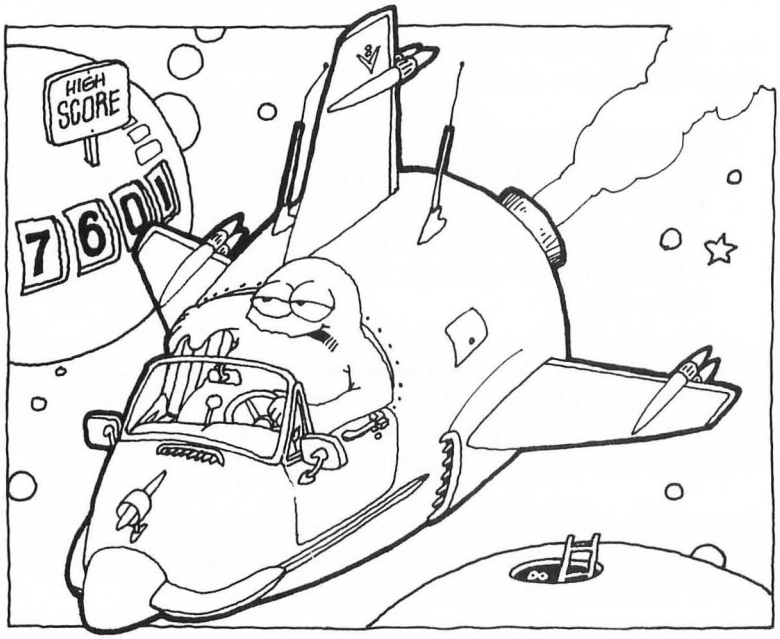
Bonus Points and Awards

Bonus points should be awarded for achieving a goal or passing a game benchmark. I like to reward players by giving them additional character lives after they've accumulated a certain number of points. This lets them continue to play. It also makes me feel like Super-Wacko!

If an arcade game player "can't get no satisfaction," he or she won't continue to play. Reward your players by giving them additional character lives, or simply award extra points. Positive reinforcement gives your players satisfaction.

High Score

Try to include both "high score" and "game score" readouts on your display. The high score gives the player a target to beat, and will encourage him or her to play your game again.



Movin' On

Now, for the moment you've been waiting for. After you flip this page you'll enter the exciting world of arcade game programming! Keep all the elements we've just discussed in the back of your mind as I reveal all my game design secrets.

Stop Before You Flip

STOP! Before you move onto the next exciting chapter, type this short program into your computer and RUN it:

```
10 GRAPHICS 18:POKE 712,128:POKE 755,5
20 POSITION 5,3:PRINT #6;"WELCOME TO"
30 POSITION 2,5:PRINT #6;"THE TOPSY-TURVY"
   :POSITION 6,7:PRINT #6;"WORLD OF"
   :POSITION 6,9
35 PRINT #6;"dr";CHR$(14);"wacko"
40 IF PEEK(53279) = 5 THEN POKE 755,5:POKE 712,128
50 IF PEEK(53279) = 6 THEN POKE 755,1:POKE 712,99
60 GOTO 40
```

Press the START and SELECT buttons to see the results. Now that I've gotten that out of my system . . .

2
Graphics Modes, COLOR and PLOT Graphics, and Lots of Other Great Stuff

This chapter is the most *important*, bar none, in this entire fantastic book! Using the elements that you'll learn here will send you on your way toward developing the arcade game of your demented dreams.

Since your fingertips are warmed up, and you're still mesmerized by my superturnemupsidedown program, I'd like you to experience more programming elegance. But, this time there *is* a method to my madness.

The programming used in these three arcade game screens will seem like child's play, once you've finished this chapter.

SUPER BREAKOUT™

```
 5 . SUPER BREAKOUT
10 GRAPHICS 23
20 COLOR 2
30 FOR A=0 TO 5
40 PLOT 29+A,80
50 DRAWTO 29+A,1+A
60 DRAWTO 130-A,1+A
70 DRAWTO 130-A,80
80 NEXT A
```

```
 90 FOR A=0 TO 1
100 COLOR A*2,1
110 FOR B=0 TO 10
120 PLOT 36,15*A+15+B
130 DRAWTO 124,15*A+15+B
140 NEXT B
150 NEXT A
160 COLOR 0
170 FOR A=35 TO 124 STEP 10
180 FOR B=0 TO 1
190 PLOT A,15*B+15
200 DRAWTO A,27*B+27
210 NEXT B
220 NEXT A
230 FOR A=14 TO 25 STEP 3
240 FOR B=0 TO 1
250 PLOT 35,A+B*15
260 DRAWTO 124,A+B*15
270 NEXT B
280 NEXT A
1000 GOTO 1000
```

TRON™

```
  5 . TRON
 10 GRAPHICS 21
 20 COLOR 3
 30 FOR A=0 TO 10
 40 FOR B=0 TO 7
 50 FOR C=0 TO 5
 60 PLOT A*7+C+2,B*6
 70 DRAWTO A*7+C+2,B*6+4
 80 NEXT C
 90 NEXT B
100 NEXT A
110 COLOR 2
120 PLOT 2,0
130 DRAWTO 77,0
140 DRAWTO 77,46
150 DRAWTO 2,46
160 DRAWTO 2,0
170 POKE 712,128
1000 GOTO 1000
```

MAZECRAZE™

```
  5 . MAZECRAZE
 10 GRAPHICS 19
 20 COLOR 1
 30 FOR A=0 TO 4
 40 PLOT A*2,A*2
 50 DRAWTO 39-A*2,A*2
 60 DRAWTO 39-A*2,23-A*2
 70 DRAWTO A*2,23-A*2
 80 DRAWTO A*2,A*2
 90 NEXT A
100 COLOR 0
110 PLOT 19,2:DRAWTO 19,21
120 PLOT 20,2:DRAWTO 20,21
130 PLOT 2,11:DRAWTO 37,11
140 PLOT 2,12:DRAWTO 37,12
150 COLOR 2
160 PLOT 10,10
170 DRAWTO 29,10
180 DRAWTO 29,13
190 DRAWTO 10,13
200 DRAWTO 10,10
210 COLOR 3
220 PLOT 11,11
230 DRAWTO 28,11
240 PLOT 11,12
250 DRAWTO 28,12
1000 GOTO 1000
```

Now that you've been "blown away" by these three familiar arcade screens, it's time to show you how you can blow *them* away!

A Thespians Life for Me?

AHH! IT'S EXCITING TO BE ON THE STAGE AGAIN! MY REALITIES ARE ILLUSIONS, ILLUSIONS ARE REALITIES, AND I GET THE CHANCE TO PERFORM, BE APPLAUDED, BOOED, HISSED, AND GET DOWNLOADED!

During my illustrious career as thespian (what?), director, and playwright, I have starred in, directed, and written arcade games that have appeared on computer stages in some of the world's most cosmopolitan centers, like Peoria and Baffin Island. Some of my finest arcade games have even appeared on the silver screen. My sing-along with the bouncing ball to the words of "Moon, Spoon on the Lagoon" was a smashing hit in nineteen-ought-two.

I know that you're anxious to put all your wonderfully imaginative arcade game ideas on the stage so you can bask in the sunshine of sucess and fame, like me—so stay weird, buckle your seat belts, and here we go!

Your Atari's Screen

Your Atari's screen, not the local Bijou, is the stage setting and backdrop for all your arcade games. But before you can astound the peanut gallery with your genius you'll have to select the proper scenery, then do some set design and construction.

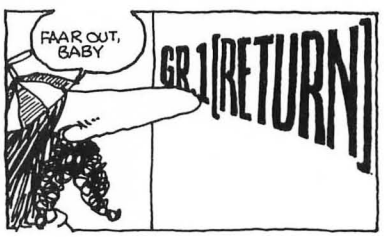
I always get a kick out of writing games that play on the Atari. This great computer offers so many different stage settings (called graphics modes) that I can always find the right one for *any* arcade game I've conceived—the weirder, the better!

Setting The Stage: Atari's Graphics Modes

Constructing a stage set and backdrop for your arcade game takes a lot of backbreaking work, so to make things easier, you first need to get acquainted with the Atari computer's world-famous graphics modes.

The first pearl of knowledge I'm going to impart is this: To select a particular graphics mode, use the BASIC command: GRAPHICS (or its abbreviated form: GR.). For example, this short, direct command selects and displays graphics mode 1:

GRAPHICS 1 <RETURN>



Since a picture is worth a megabyte of words, check out your screen after you type in and RUN the following program. It shows you all the "stage settings" your Atari computer offers. (Wacko note: Your Atari computer may have as many as *twelve* graphics modes. To keep things simple, let's only discuss and use graphics modes 0 through 8. Not that I'm lazy. It's just that most BASIC arcade games don't use modes 8 through 11 because each of these modes consumes 7900 bytes of memory—which is a lot of valuable memory that can be used for making game players such as bouncing fuzzballs stick to walls.)

Dr. Wacko's World Renowned Selecting the Stage Program

```
10 FOR X=0 TO 8
20 IF X=0 THEN GRAPHICS X:POKE 752,1:POSITION
   13,11:PRINT "GRAPHICS ";X:X=1
30 FOR A=1 TO 100:NEXT A
40 IF PEEK(53279)<>6 THEN GOTO 40
50 GRAPHICS X:POKE 752,1:PRINT :PRINT
   CHR$(127);CHR$(127);"GRAPHICS ";X:POSITION
   5,5:PRINT #6;"GRAPHICS ";X
60 FOR A=1 TO 100:NEXT A
70 IF PEEK(53279)<>6 THEN GOTO 70
80 NEXT X:GOTO 10
```

After you RUN this program, just press START to view each of the graphics modes. (SAVE this terrific program to disk or cassette.)

Vive la Difference

As you flip through each of the nine graphics modes presented in this program you'll notice a few important differences in each screen.

Text modes: The first three screens (graphics modes 0, 1, and 2) are known as the text modes and are normally used (you guessed it!) to display text.

Pixel modes: The remaining six screens (graphics modes 3 through 8) display strips of colored squares in place of text. Each colored square is called a pixel.

As you progress through this astounding book, you'll learn how to use both text and pixel modes to design your arcade game's stage.

The Thin Blue Strip

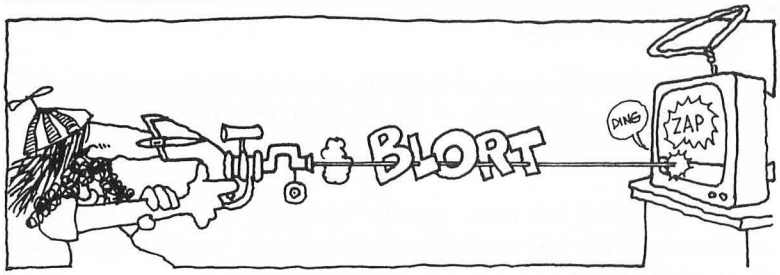
A thin blue strip runs mysteriously across the bottom of graphics modes 1 through 8. (In graphics mode 8 the blue strip is hard to see because I'm nearsighted, and the entire screen is the same blue color as the strip—but it's there. Trust me.)

This blue strip is called the *text window*. It's used to display standard letters, numbers and symbols.

You can use this area to display information such as rate of descent in a Lunar Lander game, bearing to target in a battleship gunnery game, or "Earth calling Dr. Wacko" in a Come Back to Reality game.

You enter text into this window with the PRINT command. Here's an example:

```
10 GRAPHICS 2:POKE 752,1
20 PRINT "Earth calling Dr. Wacko. . ."
```



If your game doesn't use the text window, you can destroy it—*ZAP*—and create a full screen display by adding 16 to the graphics designation number. Here's how to get rid of that thin blue strip: To change graphics mode 1 from a split-screen to a full screen display, use the command GRAPHICS 17 (1 + 16 = 17). Try this two-line program:

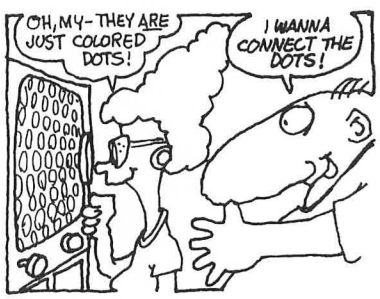
```
10 GR. 17
20 GOTO 20
```

You've eliminated that pesky blue strip, and the screen's grown! Do you remember the Selecting-the-Stage program you just SAVEd? (How soon we forget.) Well, *bring it back!* Cleverly modify the program by deleting line 50 and replacing it with:

```
50 GRAPHICS X + 16:POKE 712,X + 10 * INT(RND(0) * 15)
```

Now press the START key and flip through each graphics mode without having to watch that obnoxious thin blue strip run across the bottom of your screen. I've identified each full-screen graphics mode by taking its fingerprints and assigning it a different random color. Because I'm really not too good at this sort of thing, sometimes this color will be black. Don't panic! The next time through, it will have another color—I hope.

The Wacko Unified Hole Theory: Columns, Rows, and Coordinates



Take a magnifying glass (or squint a lot) and look at the color cover of this book.

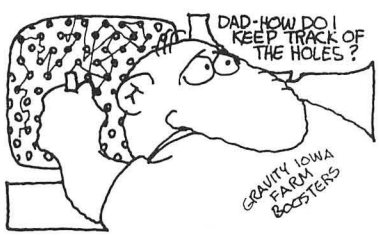
Right, Ms. Peeky. And graphics are displayed on your computer's screen in the exactly the same way—each graphics mode contains hundreds of "holes" (pixels) waiting to be filled in with color.

Resolution

Take a look at the Screen Size column of Table 2.1. The screen size for graphics mode 0 is 40 columns by 24 rows (40X24). The screen size for full screen graphics mode 3 + 16 is also 40X24. Even though one mode displays text and the other displays pixels, *both of these graphics screens have the same resolution* (number of holes per screen).

Low-resolution graphics modes 0 and 3 + 16 each have a total of 960 (40X24) empty holes waiting to be filled. *High-resolution* mode 8 has 69120 (360X192) teeny-weeny holes waiting to be filled in with color.

Coordinates



How do you keep track of all those holes? It's simple! Each hole (or pixel) is assigned its own two-number location called a *coordinate*. Here's how this nifty system works:

The numbers across the top of the illustration assign a value to each column; these are called X locations. The numbers down the illustration's side assign a value to each row; these are called Y locations. *Each pixel is identified by its two-number location coordinate.* The pixel's column location is *always* stated first, followed by a comma, then the pixel's row location, like this: X,Y.

Snidely's filled in the holes at locations 15,10 (X,Y) and 24,15 (X,Y) to show you this simple concept.

Loads of Modes: The COLOR Statement, Color Register POKES, PLOT, POSITION, DRAWTO, and LOCATE

Selecting the right graphics mode for an arcade game requires some artistic pizzazz (or lots of pizza!). I hold my thumb up in front of the screen, like Van Gogh appraising his canvas, and make comments like: "Ahh, ooh, ugh, and wow, what a nice thumb!"— until I've choosen the right graphics mode.

This method works great for artistic types like me. But will it work for you? Even if you are an accomplished computer artist with a "great" thumb, you will enjoy this fascinating section. It's loaded with some real "artistic" tricks of the trade.

Table 2.1 summarizes all you'll need to know about graphics modes, and if that's not enough, I'll give you the complete run-down on each graphics mode plus some very exciting graphics tricks.

	GRAPHICS MODE	DISPLAY TYPE	AVAILABLE COLOR	SCREEN SIZE (COLUMNS x ROWS)	COLOR REGISTER NUMBER(S)			COLOR, REGISTER NUMBER AND POKE	MEMORY USED (BYTES)
					CHARACTERS OR PIXELS	BACKGROUND	BORDER		
TEXT MODES	0	STANDARD TEXT	1 color & VARIABLE CHARACTER LUMINANCE	40 x 24	1(VARIABLE LUMINANCE ONLY)	2 (BLUE)	4 (BLACK)	—	993
	1	DOUBLE-WIDTH TEXT	5	20x20 (SPLIT) 20x24 (FULL)	0, 1, 2, 3	4	4	(SEE TABLE 2.2)	513
	2	DOUBLE-WIDTH DOUBLE-HEIGHT TEXT	5	20 x 10 (SPLIT) 20 x 12 (FULL)	0, 1, 2, 3	4	4	(SEE TABLE 2.2)	261
PIXEL MODES	3	FOUR COLOR GRAPHICS	4	40x20 (SPLIT) 40x24 (FULL)	0, 1, 2	4	4	COLOR 0, REGISTER 4 POKE 712,0 COLOR 1 REGISTER 0 POKE 708,40 COLOR 2 REGISTER 1 POKE 709,202 COLOR 3 REGISTER 2 POKE 710,148	273
	5		4	80 x 40 (SPLIT) 80 x 48 (FULL)	0, 1, 2	4	4		1017
	7		4	160 x 80 (SPLIT) 160 x 96 (FULL)	0, 1, 2	4	4		3945
	4	TWO COLOR GRAPHICS	2	80x40 (SPLIT) 80x48 (FULL)	0	4	4	COLOR 0 REGISTER 4 POKE 712,0	537
	6		2	160x80 (SPLIT) 160 x 96 (FULL)	0	4	4	COLOR 1 REGISTER 0 POKE 708,40	2025
	8	HIGH RESOLUTION GRAPHICS	1 color & VARIABLE CHARACTER LUMINANCE	360x160 (SPLIT) 360 x 192 (FULL)	1(VARIABLE LUMINANCE AND PHASE SHIFT color)	2	4	COLOR 0 REGISTER 2 POKE 710,148 COLOR 1 REGISTER 1 POKE 709,202	7900

TABLE 2·1

Pixel Modes 3 through 8: Using the COLOR and PLOT Statements

Take a look at Table 2.1 and you'll see that the pixel modes are separated into three groups:

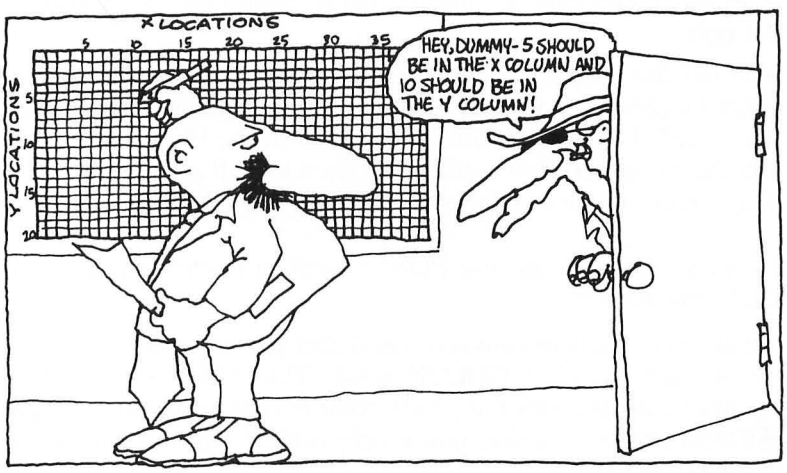
- Four-color graphics
- Two-color graphics
- High-resolution graphics

Using the COLOR statement in these modes is real easy. I'll use it in graphics mode 3 to give you the idea.

Graphics mode 3 or 19 (3 + 16) is a four-color graphics mode. This simply means that you have one background color and three pixel colors to work with. The COLOR and PLOT statements go together like R2-D2 and 3-CPO—they're inseparable. In graphics mode 3, you use COLOR 0 to "paint" the background of your playfield and COLOR's 1,2, and 3 to "draw" your playfield in vibrant hues!

The PLOT command fills a pixel on your screen with the color of your choice, at the coordinates of your choice. COLOR and PLOT are used together like this:

COLOR 1:PLOT 5,10



Here's a short program that PLOTs an orange pixel (COLOR 1) on your screen in graphics mode 3:

COLOR and PLOT

```
10 GRAPHICS 3
20 COLOR1
30 PLOT 0,0
```

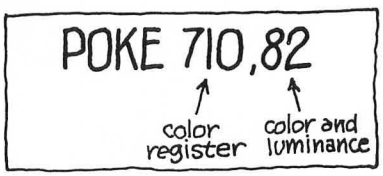
Ooops—replace line 20 first with COLOR 2, then with COLOR 3. COLOR 2 appears as light green and COLOR 3 as a small blue box—amazing! You guessed it, the four colors are preset:

COLOR 0—Black (background)
COLOR 1—Orange
COLOR 2—Light Green
COLOR 3—Blue

BUT, DAD - HOW DO I CHANGE ALL THOSE COLORS?
HAVE NO FEAR, WACKO'S HERE - WITH POKE TO THE RESCUE!
?
member ECHO VALLEY Y.OO.+DAYCARE CENTER

Have no fear, Dr. Wacko's here with POKE to the rescue!

A POKE statement's got two numbers. Study the illustration to the right.



You use the POKE statement to stuff information directly into a special location inside your Atari computer's memory. This location is called a *register.* The first number in the POKE statement tells the computer what register you want to stuff information into. The second number is the stuff that you're POKEing.

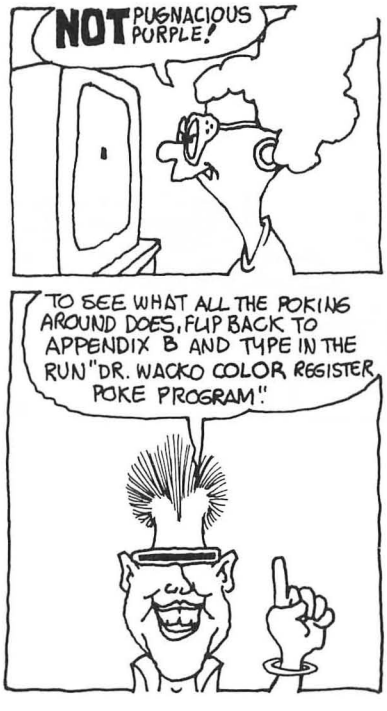
Listed to the right are the POKEs used to control each of the COLORs available.

By adding a number between 0 and 255 you can change the color of each PLOTed COLOR pixel. The added POKE color number changes both the pixel's color and its *luminance*. To get different shades of color, you simply add or subtract 2 from the color number.

	POKE REGISTER NUMBER	USE COLOR
GRAPHICS 3,5,7 (four color)	708	1
	709	2
	710	3
	712	4
GRAPHICS 4,6 (two color)	708	1
	712	0
GRAPHICS 8 (one color, two-luminances)	709	1
	712	-
	710	0

Now, while you are still dazzled by all those vibrant colors let's do some more programming. Clear your screen by typing in NEW then type in the COLOR and PLOT program again and change line 10 to read:

10 GR. 3:POKE 708,99


NOT PUGNACIOUS PURPLE!
TO SEE WHAT ALL THE POKING AROUND DOES, FLIP BACK TO APPENDIX B AND TYPE IN THE RUN"DR. WACKO COLOR REGISTER POKE PROGRAM!"

Now, when you RUN your new COLOR and PLOT program you'll see that the little dot has change to *pugnacious purple!*

Go wacko with this short program. By POKEing 708 with any number between 0 and 255 you can change the color of the pixel that you've PLOTed using COLOR 1. Experiment with other COLOR statements and registers. Replace the COLOR statement in line 20. Make it COLOR 2, and change line 10 to read:

10 GR. 3:POKE 709,195

Play with these concepts and refer to the table opposite until you've got a good grasp of how the different color POKEs control each PLOTed COLOR.

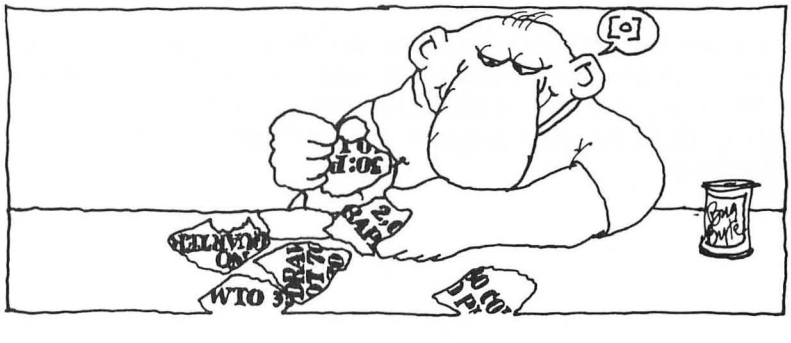
DRAWTO the Stage

Now that you're all wacked-out on color, I'll show you how to use the DRAWTO command to draw lines on the screen. But before you use DRAWTO you'll have to PLOT a beginning coordinate. DRAWTO is *always* used with PLOT like this:

PLOT X,Y:DRAWTO X,Y

Put the COLOR and PLOT program back together again and add this line:

40 DRAWTO 39,19



Again, fool around with this new program, then we'll mess it up some more.

O.K.?

Now, using the same program, add this line 15:

15 POKE 708,99

And change line 40 to read:

40 DRAWTO 0,19:DRAWTO 39,19:DRAWTO 39,0:DRAWTO 1,0

Now, when you RUN your modified program, a strip of purple borders the edge of the screen. It's *pugnacious purple* again because we've POKEd 708, COLOR 1's register, with 99. Remember?

Line 40 shows that you can "chain" DRAWTO commands one after another. You can draw any arcade playing field imaginable by combining these PLOT, DRAWTO, and COLOR techniques!

Steppin' through Mazecraze

Now, armed with your knowledge of graphics modes and commands, flip back to the Mazecraze program and we'll step through it.

In line 10, I've selected "windowless" graphics mode 19 (3 + 16).

To speed up the drawing process I often use the handy FOR/NEXT loop. Lines 20 through 90 of the Mazecraze program quickly draw five COLOR 1 lines of decreasing lengths from the top to the bottom of the screen.

Here are lines 10 through 90 of the Mazecraze program. I've slowed down the drawing process by adding a pause FOR/NEXT loop in line 85 so you can watch the screen being drawn. RUN this short program and you'll get the picture—or a headache!

Mazecraze FOR/NEXT loop Demo

```
 10 GRAPHICS 19
 20 COLOR 1
 30 FOR A=0 TO 4
 40 PLOT A*2,A*2
 50 DRAWTO 39-A*2,A*2
 60 DRAWTO 39-A*2,23-A*2
 70 DRAWTO A*2,23-A*2
 80 DRAWTO A*2,A*2
 85 FOR PAUSE=0 TO 200:NEXT PAUSE
 90 NEXT A
100 GOTO 100
```

The balance of the program, lines 100 through 250, is broken into sections headed by a COLOR statement and followed by a bunch of PLOT and DRAWTO statements. COLOR 0, the background color, is used to "cut away" sections of the lines that were first drawn. COLORs 2 and 3 are used to draw the design in the center of the screen.

Now that you know what you are doing (I wish I could say the same for me), use your new skills to modify the three professional arcade game playing field examples. Change their colors; put them in different graphics modes; change their design. Go completly wacko—really mess 'em up!

Action in the Text Modes!

Graphics Mode 0

Graphics mode 0 is usually used to display text, but stick with Dr. Wacko, kid, and I'll show you how to design arcade games in this mode (if you'll be a little patient).

Just so you'll recognize it when you see it, graphics mode 0 is that blue display with a black background that you see when you turn on your computer.

Here's some basic information about graphics 0:

- Normally used to display text
- Screen size: 40 characters across and 24 characters down (38 X 24)

- Colors: 1 color; 2 degrees of brightness. (POKE 710,XX to change the screen's color. POKE 712,XX to change the screen's border. POKE 708,XX to change the text character's brightness.)
- Command: GRAPHICS 0

Graphics Modes 1 and 2: El Biggo Texteroonio

Graphics modes 1 and 2 are also *normally* used to display text. But not after the famous Dr. C. Wacko gets his hands on them. Once you've finished this book you'll know how to use these valuable modes in some pretty subnormal ways. To display entire arcade games, for instance!

Trip between graphics modes 0, 1, and 2 in my Selecting-the-Stage program and once you get back up you'll see that letters and symbols printed in graphics mode 1 are twice the width of those printed in graphics 0, but are the same height.

Letters and symbols printed in graphics mode 2 are the same width as graphics mode 1 characters, but are two times greater in height.

Here's the scoop on graphics modes 1 and 2:

- Screen size: Graphics Mode 1
  Split Screen: 20 characters across by 20 characters down (20X20)
  Full Screen: 20 characters across by 24 characters down (20X24)
- Screen size: Graphics Mode 2
  Split Screen: 20 characters across by 10 characters down (20X10)
  Full Screen: 20 characters across by 12 characters down (20X12)
- COLORS: 5 (4 character colors and 1 background color)

Plotting in the Text Modes

Many of my most spectacular games are presented in the text modes. I often use either graphics mode 1 or 2 because each offers *five colors*. I've even used graphics mode 0 a few times.

In the next chapter you'll learn how to change the letters and symbols that normally appear in the text modes to any shape you can imagine. When you do you'll be able to develop text mode arcade games that are loaded with color and pizzazz!

Now I'll show you how to PLOT in these text modes so you'll be ready to design game screens with the weird shapes I'm sure you'll soon be creating.

Here's a short, but brilliant example:

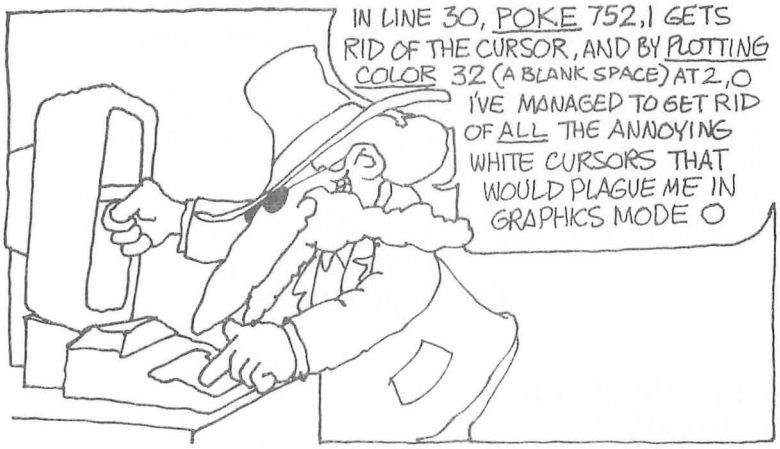
Text Mode Plotting

```
 10 GRAPHICS 1
 20 GOTO 40
 30 POKE 752,1:COLOR 32:PLOT 2,0
 40 COLOR 87
 50 PLOT 0,0
 60 COLOR 97
 70 PLOT 1,1
 80 COLOR 227
 90 PLOT 2,2
100 COLOR 203
110 PLOT 3,3
120 COLOR 111
130 PLOT 4,4
```

First RUN this program as is and you'll see "WACKO" printed in brilliant colors.

Now change line 10 to: 10 GRAPHICS 2, and RUN it again—BIGGER letters!

Now try it in graphics mode 0 by deleting line 20, then changing line 10 to: 10 GRAPHICS 0.


IN LINE 30, POKE 752,1 GETS RID OF THE CURSOR, AND BY PLOTTING COLOR 32 (A BLANK SPACE) AT 2,0 I'VE MANAGED TO GET RID OF ALL THE ANNOYING WHITE CURSORS THAT WOULD PLAGUE ME IN GRAPHICS MODE 0

What's happening here? The number you put behind the COLOR statement, like COLOR 87, is the ATASCII value of the letter or symbol that will be printed at the PLOT coordinates! In line 40 of my brilliant program, COLOR 87 will print the first letter of my name, W, at coordinates 0,0.

Refer to the ATASCII Chart on page 189 to modify this program and put your name on the silver screen!

Using POSITION

Oh, before I forget (and I never do that!), you can also use PRINT (in GR.0) or PRINT #6; (in GR.1 or 2) with the POSITION statement to place characters on the screen. Try this one-liner:

```
10 GRAPHICS 1:POSITION 5,5: PRINT #6;CHR$(87);
   CHR$(97); CHR$(227); CHR$(203); CHR$(111)
```

Using PRINT #6 to Display El Biggo Texteroonio

In both graphics modes 1 and 2, uppercase, lowercase, and inverse video letters, numbers, and symbols normally appear on your screen as *upper-case* text. But, here's the exciting part. Each has *its own distinct color!*

Type in and RUN this short program, and you'll get the picture. (Underlined letters indicate inverse text. Press the key with the Atari symbol to enter these characters. Press it again to return to normal text.)

Colorful Letters

```
10 GR.2
20 POSITION 3,5
30 PRINT #6;"El bIggO tEXt"
```

Isn't that shocking! Now let's have some fun.

First, change line 10 to read GRAPHICS 1 and RUN it again. Neato!

O.K., now we come to the devious part. I'm going to show you how to *mess up all those colors!*

Let your fingers do the walking to Table 2.2. There yet?

Let's give it a try. Type: POKE 708,99 <RETURN>. All the letters that were entered as standard uppercase letters—the E in "EL", the I in "BIGGO", and the E in "TEXT" have all miraculously changed color. They're all putrid purple! Eeeyuk!

POKE other numbers into location 708 by changing the number after 708. Try combinations like 708,23; 708,185; and so on, so forth; and to wit.

Figure out that chart yet? You did! Then you realize that you can change the other letter's colors in the same manner.

Here's the explanation Slow Poke gave to me:

POKE 708,XX changes the color of *standard uppercase* letters.
POKE 709,XX changes the color of *standard lowercase* letters.
POKE 710,XX changes the color of *inverse uppercase* letters.
POKE 711,XX changes the color of *inverse lowercase* letters.

One additional thought. You can also change the *background color* from black to any color your heart desires by POKEing 712 with any number between 1 and 255 (0 is black so don't waste your time entering it). POKE 712,185 turns the background to my favorite color, swampwater green.

TABLE 2.2 Color in Graphics Modes 1 and 2

ATASCII Value for Color Register 0	1	2	3	Character	ATASCII Value for Color Register 0	1	2	3	Character
32*	0	160	128		50	18	178	146	2
33	1	161	129	!	51	19	179	147	3
34	2	162	130	"	52	20	180	148	4
35	3	163	131	#	53	21	181	149	5
36	4	164	132	$	54	22	182	150	6
37	5	165	133	%	55	23	183	151	7
38	6	166	134	&	56	24	184	152	8
39	7	167	135	'	57	25	185	153	9
40	8	168	136	(	58	26	186	154	:
41	9	169	137	)	59	27	187	None	;
42	10	170	138	*	60	28	188	156	<
43	11	171	139	+	61	29	189	157	=
44	12	172	140	,	62	30	190	158	>
45	13	173	141	-	63	31	191	159	?
46	14	174	142	.	64	96	192	224	@
47	15	175	143	/	65	97	193	225	A
48	16	176	144	0	66	98	194	226	B
49	17	177	145	1	67	99	195	227	C

ATASCII Value for Color Register				Character	ATASCII Value for Color Register				Character
0	1	2	3		0	1	2	3	
68	100	196	228	D	82	114	210	242	R
69	101	197	229	E	83	115	211	243	S
70	102	198	230	F	84	116	212	244	T
71	103	199	231	G	85	117	213	245	U
72	104	200	232	H	86	118	214	246	V
73	105	201	233	I	87	119	215	247	W
74	106	202	234	J	88	120	216	248	X
75	107	203	235	K	89	121	217	249	Y
76	108	204	236	L	90	122	218	250	Z
77	109	205	237	M	91	123	219	251	[
78	110	206	238	N	92	124	220	252	\
79	111	207	239	O	93	None	221	253	]
80	112	208	240	P	94	126	222	254	^
81	113	209	241	Q	95	127	223	255	_

* 155 selects the same character and color register as value 32.

POKE 708,X FOR COLOR REGISTER 0 (uppercase)
POKE 709,X FOR COLOR REGISTER 1 (lowercase)
POKE 710,X FOR COLOR REGISTER 2 (inverse uppercase)
POKE 711,X FOR COLOR REGISTER 3 (inverse lowercase)

Well, that was real intersting, but what, you may ask (I might not answer) is so *important* about all this colorful nonsense?

How dare you! This is really, really, ad infinitum, *important stuff!*

In the next chapter, Character Graphics, I'm going to show you how to change these letters into shapes of all kinds: monsters, lollipops, kangaroos, rocket ships, Albert Einstein (?).

After you learn how to transform, the letter *A*, for example, into a reasonable facsimile of a lollipop, for example, *you'll know how to change the lollypop's color*. Just think of it, any flavor your heart desires, even swampmuck green. (Double EEeeyuk!)

That's why this stuff is really important!

LOCATE and Collisions

Chances are, you're going to want lots of *action* in your games. When a missile strikes its target you'll want to see a flash and hear an ear-splitting explosion—*BAAROOOOOM!* When a ball hits the side of a wall, you want to know it—*PING!* One way to let the other elements in your program know that a collision has occurred is to use the LOCATE statement. LOCATE is used in this format: LOCATE X,Y,Z

X is the column location of the collision, Y is the row location, and Z is value that's encountered at location X,Y. (You don't have to use Z—any variable other than X or Y will work just fine.)

In graphics modes 3 through 8 the value returned in Z is the number that follows the COLOR statement: either 0, 1, 2, or 3.

In graphics modes 0 through 2 the value returned in Z will be the ATASCII value of the character encountered. For example, if the letter A is located at coordinates 5,5 in graphics mode 0 (LOCATE 5,5,Z), the value returned in Z will be 65. (Now's a good time to look over the *ATASCII Codes, Characters and Keystrokes* chart on page 189.) Here's a short program that shows the basic LOCATE concept. There's a surprise program at the end of this chapter that uses LOCATE in an action arcade situation!

LOCATE Demo

```
 10 GRAPHICS 3
 20 COLOR 1
 30 PLOT 0,0
 40 COLOR 2
 50 PLOT 1,1
 60 COLOR 3
 70 PLOT 2,2
 80 LOCATE 0,0,A
 90 LOCATE 1,1,B
100 LOCATE 2,2,C
110 LOCATE 3,3,D
120 PRINT A,B,C,D
```

After you RUN this program three colored squares will be displayed at the upper left of the screen. The corresponding COLOR statement numbers (1, 2, 3, and 0) are LOCATEd, then printed in the text window.

COLOR 1 (register 0—orange) is plotted at 0,0.
COLOR 2 (register 1—green) is plotted at 1,1.
COLOR 3 (register 2—blue) is plotted at 2,2.
COLOR 0 (register 4—black) is the background color.

What's This? Wacko's Secret Formula?: SC = PEEK(88) + 256 * PEEK(89)

Not at all! I'm going to show you a *super fast* way to POKE stuff to and from the screen.

I'll first show you how to use this method in graphics mode 19 (3 + 16), then how to apply it to other graphics modes.

POKEing Stuff to GRAPHICS 19

In graphics mode 19, in the statement SC = PEEK(88) + 256 * PEEK(89), SC is the location of the first four pixel locations at the upper left corner of the screen—coordinates 0,0; 1,0; 2,0; and 3,0.

This program will place color-filled pixels (blue, green, orange, blue) at the top left of the screen.

```
10 GRAPHICS 19
20 SC = PEEK(88) + 256 * PEEK(89)
30 POKE SC,231
40 GOTO 40
```

Here's how this little gem works.

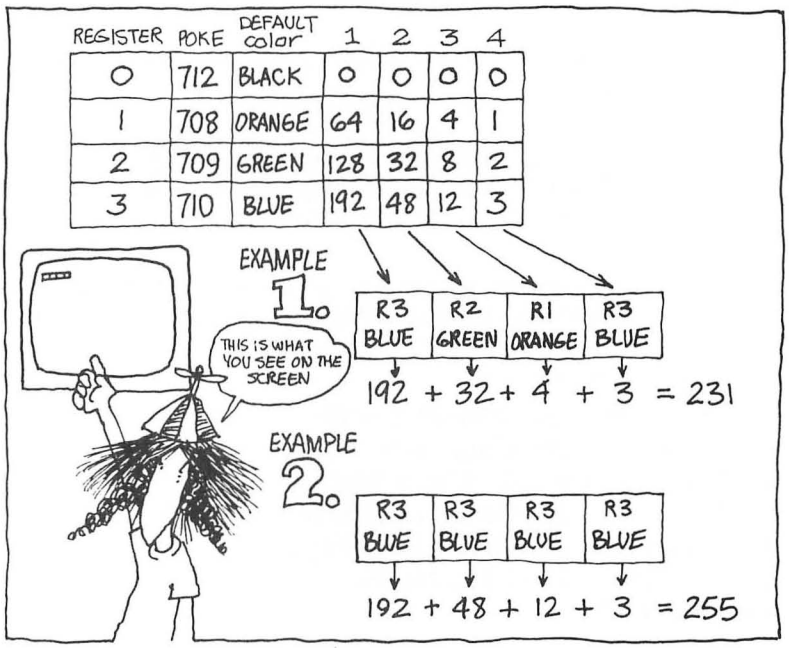
Line 20: The variable SC is assigned the value of the upper left corner of the screen.

Line 30: POKE color code 231 to the screen at the upper left corner.

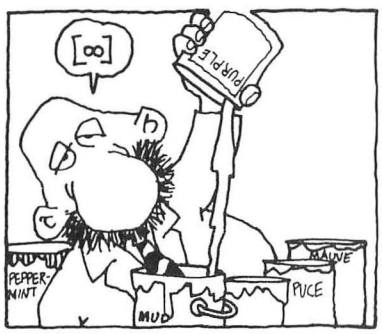
Line 40: The GOTO 40 is used to prevent the screen from returning to graphics mode 0.

What's COLOR Code 231?

Here's a handy-dandy chart that explains all!


REGISTER POKE DEFAULT color 1 2 3 4
0 712 BLACK 0 0 0 0
1 708 ORANGE 64 16 4 1
2 709 GREEN 128 32 8 2
3 710 BLUE 192 48 12 3
EXAMPLE 1
THIS IS WHAT YOU SEE ON THE SCREEN
R3 BLUE R2 GREEN R1 ORANGE R3 BLUE
192 + 32 + 4 + 3 = 231
EXAMPLE 2
R3 BLUE R3 BLUE R3 BLUE R3 BLUE
192 + 48 + 12 + 3 = 255

The number 231 is simply the addition of 192+32+4+3.



To change the color of all four pixels to blue, for example, you'd just replace code 231 with 255 (192+48+12+3).

Experiment with different color combinations; then I'll show you how to position these four pixels *anywhere* on the screen.

The screen size of graphics mode 19 is 40X24. That's 960 holes waiting to be filled with color!

In this graphics mode, four color-filled pixels are placed on the screen at a time. The coordinate (X,Y) system doesn't apply to this method. The first pixel is placed at screen position 0, the fourth pixel is placed at screen postion 4 moving from left to right across the screen.

Since four pixels are placed on the screen at a time (beginning at position 0) and the screen contains 960 holes, position 239 is the last position in which you can place a set of four colored pixels (960/4 – 1 = 239).

Replace line 30 with: 30 POKE SC+239,231.

When you RUN the program again, the four colored pixels will magically appear at the bottom *right* corner of the screen!

Use these formulas to place pixels at the left bottom in any graphics mode:

Graphics Modes 3, 5 & 7: SC + (Number of holes/4) – 1
Graphics Modes 4, 6 & 8: SC + (Number of holes/8) – 1
Graphics Modes 0, 1 & 2: SC + (Number of holes) – 1

Positioning your pixels is really easy. Here's a short program that'll place pixels in each corner of the screen, and in the middle.

```
10 GRAPHICS 19
20 SC = PEEK(88) + 256*PEEK(89)
30 POKE SC,231:. Upper Left
40 POKE SC + 9,231:. Upper Right
50 POKE SC + 230,231:. Lower Left
60 POKE SC + 239,231:. Lower Right
70 POKE SC + 114,231:. Middle
80 GOTO 80
```

Or, fill the entire screen with color!

```
10 GRAPHICS 19
20 SC = PEEK(88) + 256*PEEK(89)
30 FOR X = 0 TO 239
40 POKE SC + X,231
50 NEXT X
60 GOTO 60
```

To place pixels randomly, change line 40 to: 40 Y = INT(RND(0)*240):POKE SC + Y,231.

Now you can design arcade game playing fields without using PLOT and DRAWTO. This method is much, much, much faster!

Using SC to Retrieve Values, or, Who Needs LOCATE?

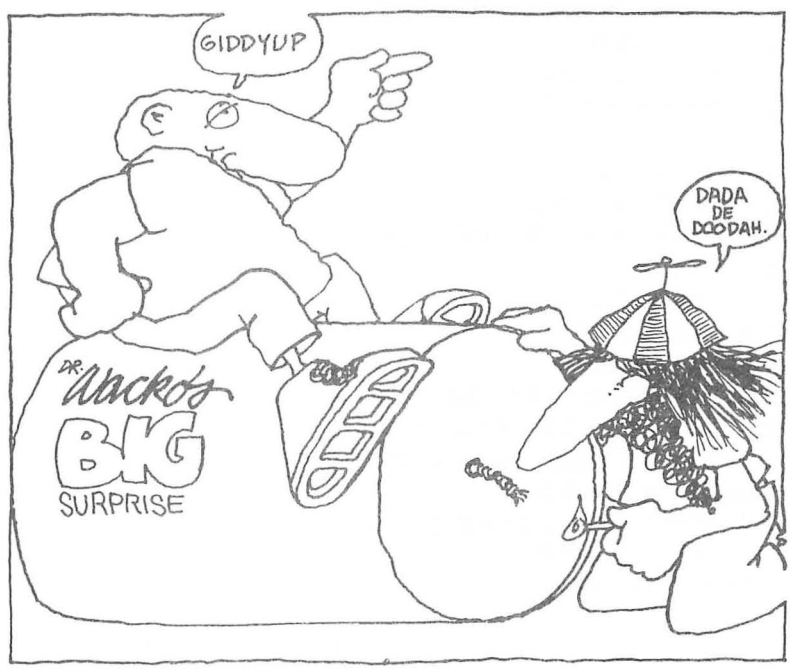
You can use this fabulous method in place of the LOCATE statement. It works super in graphics modes 0, 1, and 2 because it returns the ATASCII value of the letter on the screen. It's a little

difficult to use in the pixel modes because it will return the combined value of all the pixels at any location.

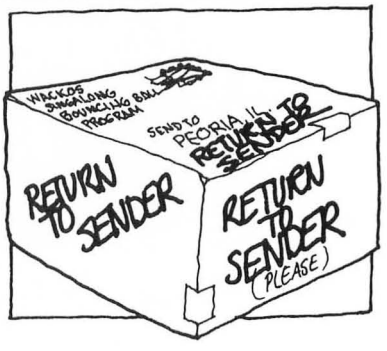
Here's how to use SC in graphics mode 0:

```
10 GRAPHICS 0
15 SC = PEEK(88) + 256*PEEK(89)
20 COLOR 65
30 PLOT 0,0
40 COLOR 66
50 PLOT 1,0
51 COLOR 67
52 PLOT 2,0
60 A = PEEK(SC)
70 B = PEEK(SC + 1)
72 C = PEEK(SC + 2)
80 PRINT :PRINT A,B,C
```

Don't go away yet! Here's a BIG surprise. . .



Wacko's Bong Program



And now, here's the grand finale bonus program I promised earlier. I've titled it Bong. It's a version of my famous "sing along with the bouncing ball" program that wowed 'em in Peoria. This superprogram uses many of the elements that you have learned in this chapter. It's also a great example of LOCATE's use in an arcade game, so without any further padew. . . here's Bong:

Bong

```
 10 GRAPHICS 19:POKE 710,15
 20 X = 1:Y = 1:DX = 1:DY = 1:XB = X:YB = Y
 30 SC = PEEK(88) + 256*PEEK(89)
 40 FOR A = 0 TO 3
 50 POKE SC + 113 + A,85:NEXT A
 60 COLOR 1:PLOT 0,0:DRAWTO 38,0
 70 PLOT 0,23:DRAWTO 39,23
 80 COLOR 2:PLOT 0,1:DRAWTO 0,23
 90 PLOT 39,0:DRAWTO 39,22
100 COLOR 0
110 PLOT XB,YB
120 XB = X:YB = Y
130 X = X + DX
140 Y = Y + DY
150 LOCATE X,Y,Z
160 IF Z<>0 THEN GOTO 210
170 COLOR 3
180 PLOT X,Y
190 POKE 77,0
200 GOTO 100
210 SOUND 0,100,10,10
220 IF Z = 1 THEN DY = -DY
230 IF Z = 2 THEN DX = -DX
240 X = XB:Y = YB
250 POKE 707 + Z,PEEK(20)
260 SOUND 0,0,0,0
270 GOTO 130
```

CONGRATULATIONS! Pat yourself on the back, have a wild and crazy party! You understand and can now *almost* design programs like this one.

I said "almost" because I will be introducing the animation elements used in Bong in chapters 3 and 4. But you've come a long way!

You now know how to select the proper graphics mode for a game like Bong, and how to use the COLOR, PLOT and DRAWTO statements to draw the playfield. You understand the infamous SC and how to use LOCATE.

Bong Explained

Here's an explanation of the portions of Bong that you should have a good handle on now.

Line 10: This line first selects graphics mode 3 + 16, then POKE's COLOR 3 with the number 15. This command (POKE 710,15) paints the bouncing ball white.

Line 20: These commands assign the ball's movement parameters, (More on movement in upcoming chapters!)

Line 30 assigns the value of the top of the screen to "SC".

Lines 40 and 50 are used to draw the horizontal bar at the center of the screen. I've used a FOR/NEXT loop to speed up the drawing process, and I've selected 85 as the color to draw with. Look at the SC chart and you'll see that 85 is the total of 64 + 16 + 4 + 1 and draws an organic orange (COLOR 1) line.

Lines 60 and 70 draw lines across the top and bottom of the screen using COLOR 1.

Lines 80 and 90 draw lines down the left and right sides of the screen using COLOR 2.

Here's the *exciting* part of this program!

Lines 150 and 160 make the Bong "action." The Z in the LOCATE statement in line 150 senses which "wall" COLOR the ball has hit. In Line 160 IF the value of Z is *not* 0, the ball has collided with one of the walls (COLOR 1 or 2) and the program jumps to line 210.*Bong!*

Line 210 lets the player know that the ball has hit one of the walls by SOUNDing off.

Line 220: The ball bounces away from a COLOR 1 wall (a horizonal wall).

Line 230: The ball bounces away from a COLOR 2 wall (a vertical wall).

Line 250: When the ball hits a COLOR 1 wall, POKE 707+Z equals 708 (707+1), the color register for COLOR 2. When the ball hits a COLOR 2 wall, POKE 707+Z equals 709 (707+2), the color register for COLOR 2.

Now, let's take a closer look at another wacko thing that's happening in line 250: POKE 707+Z,PEEK(20)

The PEEK(20) command *looks into the contents of register 20* which generates a constant stream of ever-changing numbers between 0 and 255. By continually changing the number that follows POKE 707+Z the wall changes to a different color each time it's hit by the ball.

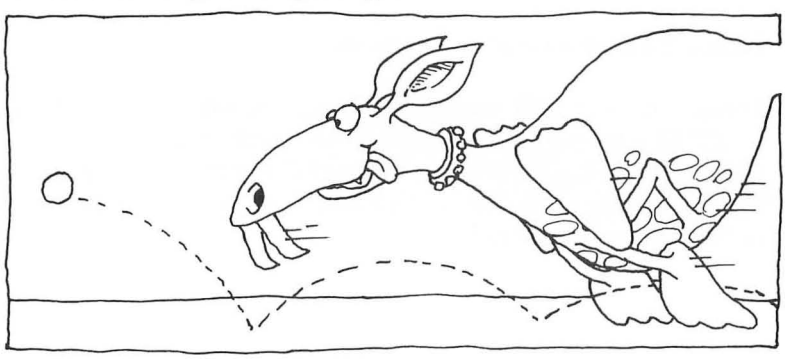
Here's a short demo program that shows how PEEK(20) works:

10 PRINT PEEK(20):GOTO 10

Here's another example:

10 GRAPHICS 3
20 POKE 712,PEEK(20):GOTO 20

Line 260 turns off the sound. Line 270 returns the program to Line 130 to begin the cycle again.



Are you ready to create a few monsters? Flip the page and enter the dank environ of my secret laboratory.

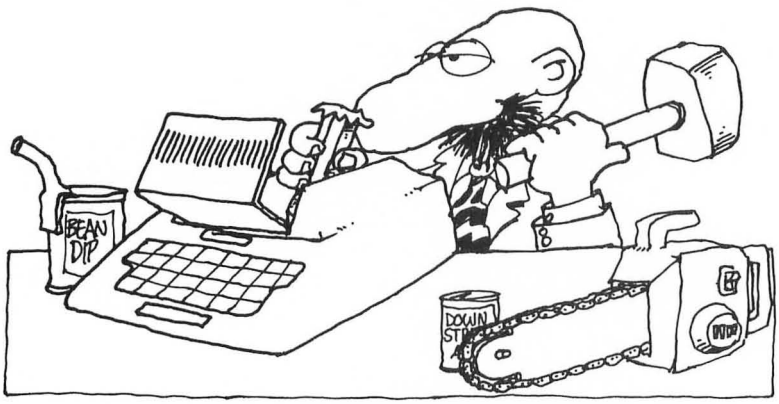
3
Character Graphics

Some people thought I had gone stark, raving mad when I announced to the world that out of tiny bytes and miniscule bits, I, the great Dr. C. Wacko, would create the *perfect arcade monster*!

After saying farewell to my students, I retreated to the dank confines of my laboratory and set down to work.

My plan was simple and direct. Learn how the "Big C" creates its characters, then use this awesome power to produce LIFE ITSELF! (Well, almost, anyway.)

I gently placed my Atari on the operating table and with scalpel in hand began my dissection. My probing took me deep into the



mysterious realm of ROM. Step by step, as my investigations continued, I unraveled *all* of Big C's hidden secrets. Now, at last, I can reveal these secrets to you.

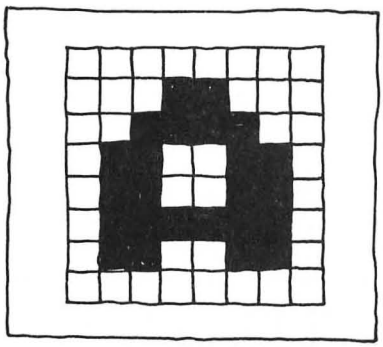
The Character Set

The letters *A* through *Z*, in upper and lower case, the numbers 0 through 9, and all the other characters printed on the Atari keyboard are all members of the character set. A complete list of characters, along with lots of helpful information about them, appears on page 189.

Your Atari prints characters to the screen after first "asking" ROM what each character looks like. I'm going to show you how to

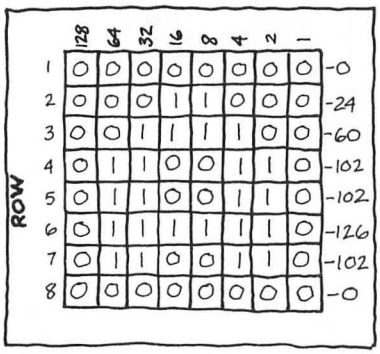
confound your Atari by defining your own custom characters. Then we'll make your Atari point to *your* character set and display your weird creations on the screen.

To do all of this weird stuff you'll have to understand how characters are defined in ROM and how the Atari knows what to print. Here's what I had to go through.



Armed with my own Atari computer, I began to experiment. I first needed to understand how each character was presented. I printed the letter A on the computer's screen, then looked at it closely through a magnifying glass. Here is what I saw:

I leaned closer. Ahaaa! The letter "A" was made up of rows of squares, forming an 8X8 grid. Each of the squares that make up the letter is actually one binary *bit* of information. Each row across contains one *byte* of information. I was getting closer to my answer. Now, all I had to do was convert this binary information into decimal numbers. The type of numbers that we humans use and *understand*.



The computer remembers the letter A as *binary numbers*, just a bunch of 0's and 1's. Shade in all the 1's and you'll begin to get the picture:

Each column of this 8X8 box has a value assigned to it. I placed these values at the top of each column to help me analyze the letter *A*.

Going across from left to right, starting with the first row, I added up the values of all the 1's. There are only 0's in the first row, so I placed a 0 to the right of the row.

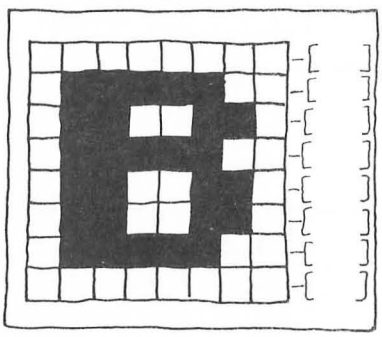
Going across the second row there is a 1 under the number 16 and another 1 under the 8, so I calculated $16 + 8 = 24$ and placed the number 24 to the right of this row.

The third row has 1's under the numbers 32, 16, 8, and 4. I added these together and placed the total —60— to the right of this row.

Continuing this process for all eight rows, I arrived at these totals for each row:

ROW 1—0
ROW 2—24
ROW 3—60
ROW 4—102
ROW 5—102
ROW 6—126
ROW 7—102
ROW 8—0

I was begining to to get the hang of it! Converting computer binary numbers into human decimal numbers was pretty easy. I was getting closer to total understanding. Soon I would be able to create the perfect arcade monster!



I've drawn the letter *B*, just like your Atari displays it. See if you can fill in the blanks next to each row with the correct decimal number.

Now is a good time to flip to page 200 and type in and SAVE the ATASCII Code program listing. If you follow the simple operating instructions, your computer will show you the decimal numbers that make up every one of Atari's characters.

I now understood how my Atari displays characters. I also knew about the list of eight decimal numbers that make up each character.

Now I had to learn where and how my Atari stores each character.

Location 57344

Digging deeper into the hidden recesses of ROM, I discovered that this cast of characters lives in a small, 1024-byte corner of Atari's memory begining at memory location 57344 and ending at location 58367.

This information didn't mean much to me until I analyzed this momentous discovery.

I decided that I would take a PEEK at the contents of each location to see if I could increase my understanding.

To do this I developed this simple program. Type it in and RUN it to experience Dr. C. Wacko's great revelation:

PEEKER

```
 10 GRAPHICS 0:POKE 752,1:POKE 710,0:? :? :?
    CHR$(127);CHR$(127);"OFFSET: 0":? :? :? :?
    :FOR X = 0 TO 1023
 20 ?CHR$(127);CHR$(127);57344 + X;CHR$(127)
    ;PEEK(57344 + X)
 30 IF INT((X + 1)/8) = (X + 1)/8 THEN POSITION
    4,18:? "PRESS START TO FLIP THROUGH
    MEMORY":GOSUB 100
 40 IF X + 1> = 1024 THEN X = 0:GOTO 10
 50 NEXT X
 60 STOP
100 IF PEEK(53279)<>6 THEN GOTO 100
110 GRAPHICS 0:POKE 77,0:POKE 710,0
    :POKE 752,1
120 ? :? :? CHR$(127);CHR$(127);"OFFSET: "
    ;X + 1:? :? :? :?
130 RETURN
```

I methodically typed the program in, carefully checking my entries every step of the way. I was ready!

I held my breath, typed RUN and hit RETURN and there it was! My screen had filled with sets of eight numbers.

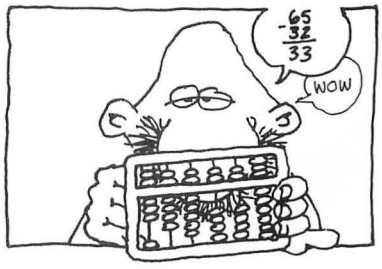
EACH SET OF EIGHT LOCATIONS AND VALUES IS ONE CHARACTER!

I anxiously pressed the START key and flipped through all of the character set locations (57344 to 58367). The second time through, I paused to examine OFFSET: 264.

I FOUND THE LETTER *A*!

And if the ATASCII decimal code is 96 to 127, MULTIPLY THE DECIMAL CODE BY 8 TO ARRIVE AT THE OFFSET NUMBER.

I took out my pocket calculator and checked my conversion table. Of course, I used the letter *A* as my example. Here's how it worked out:



- The Decimal code for the letter *A* is 65.
- Since 65 is between 32 and 95, I first subtracted 32 from 65 (65 – 32 = 33).
- Next, following my ingenious formula, I multiplied 33 by 8 to arrive at *A*'s offset number (33X 8 = 264).

264! (Thank goodness.)

MY FORMULA WORKED!

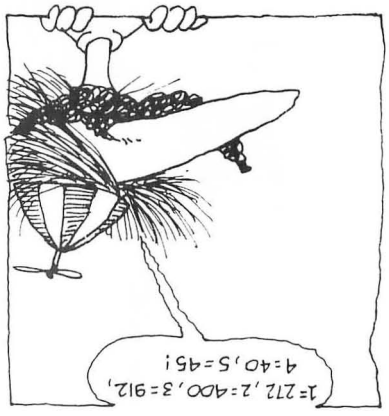
If you'd like to experiment with these conversions, here are a few examples.

What is the Offset number for the following characters?

1. B
2. R
3. r
4. %
5.)

I had discovered:

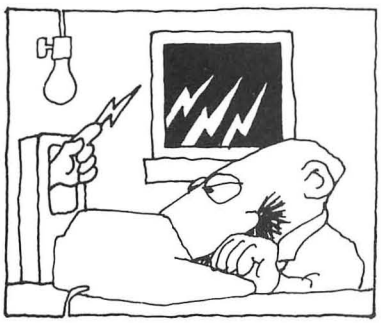
- THAT THE COMPUTER DISPLAYS EACH CHARACTER AS A BUNCH OF SQUARES
- THAT EACH SQUARE IS A BIT IN MEMORY AND A ROW OF THESE SQUARES IS A BYTE
- THAT THE COMPUTER REMEMBERS EACH LETTERS AS A BUNCH OF 0's and 1's (BINARY NUMBERS) (I ALSO LEARNED HOW TO CONVERT BINARY NUMBERS TO DECIMAL NUMBERS.)
- THAT THE COMPUTER STORES EACH CHARACTER IN GROUPS OF 8, WITHIN LOCATIONS 57344 TO 58367



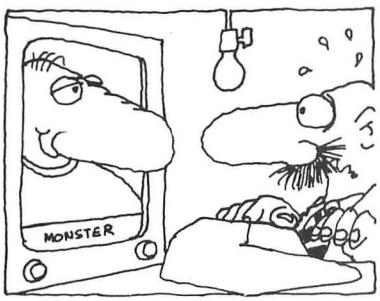
- THAT EACH CHARACTER IS MADE UP OF 8 DECIMAL NUMBERS
- THAT WHEN YOU ADD THE OFFSET NUMBER TO 57344 YOU ARRIVE AT THE LOCATION WHERE A CHARACTER'S DEFINITION BEGINS
- HOW TO CONVERT A CHARACTER'S DECIMAL CODE TO AN OFFSET NUMBER

I was ready to create my own creatures!

The Act of Creation



Bright fingers of energy danced from the small probe set next to my computer. The storm was at peak force as I began work on my first creation.



Fade to classroom. . .

Character Graphics Programming Techniques

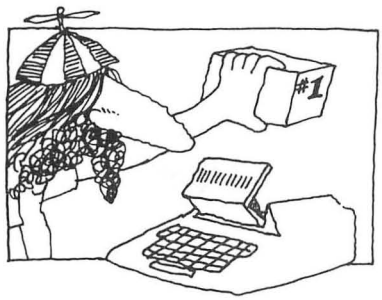
Now that you're here in my classroom, I can finally reveal how on that fateful day I created my first character. Actually, it didn't turn out exactly as planned. I had created a *MONSTER*—a veritable Wackenstein!

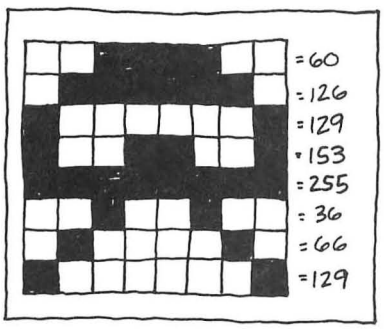
You see before you a short program listing entitled Building Block #1. This program lays the foundation for all your future work in character graphics. It also incorporates many of the concepts you've learned in the previous chapter. Best of all, when you RUN this program, part of the fabulous monster I created so many years ago will appear on your TV's monitor. (The whole, living, snorting monster will cavort for you in chapter 4.)

During the balance of this chapter I will add to and improve Building Block #1. You will continuously learn new and exciting techniques, so lets get started!

Building Block #1

```
 10. *** BUILDING BLOCK #1: CHARACTER
     REDEFINITION ***
 20 CHARACTERS = 1
 22 .
 24 .
 30 START = (PEEK(742) - 2)*256
 32 .
 40 POKE 559,0
 42 .
 50 FOR X = 0 TO 511
 60 POKE START + X,PEEK(57344 + X)
 70 NEXT X
 72 .
 80 FOR LOCATION = 0 TO CHARACTERS - 1
 90 FOR BYTE = 0 TO 7
100 READ SHAPE
110  POKE START + (LOCATION)*8
      + BYTE + 264,SHAPE
120 NEXT BYTE
130 NEXT LOCATION
132 .
140 POKE 559,34
142 .
150 GRAPHICS 2:POKE 756,START/256
152 .
160 FOR LOCATION = 0 TO CHARACTERS - 1
170 PRINT #6;CHR$(LOCATION + 65);
180 NEXT LOCATION
182 .
184 .
1000 DATA 60,126,129,153,255,36,66,129
```





Building Block #1 Explained

I'll wait while you enter Building Block #1 into your Atari's memory.

Finished? Great! Now RUN it. A *MONSTER*, right in the upper left hand corner of your screen!

Before doing any programming, I first drew the monster on 8X8 gridded paper. Here's how Wackenstein first looked:

I've listed the decimal value of each row to the right of my drawing. These numbers are now placed, in DATA form, in line 1000 of the program. I've used line 1000 so we can expand the program later.

20 CHARACTERS = 1

The variable CHARACTERS is used to keep track of the number of characters we are redefining. This program only makes one Wackenstein. Thank goodness! If we redefine one character, CHARACTERS = 1; if we redefine two characters, CHARACTERS = 2; if we redefine three characters, CHARACTERS = 3, and so on into insanity. (Or until your computer runs out of memory.)

30 START = (PEEK(742) – 2)*256

PEEKing location 742 and multiplying by 256 shows us where the top of user-available memory is. We come down from the top of memory by 512 bytes (2*256) to make room for our new character set. START represents the beginning of this new location.

An importaint point. This example is displayed in graphics mode 2. Graphics modes 1 and 2 use only the uppercase letters of the alphabet and some symbols (64 characters in all), so you only have to reserve 512 bytes to accommodate them.

Graphics mode 0 uses the entire character set (128 characters). *Achtung!* You must reserve 1024 bytes to make room for all these characters. Just substitue a 4 for the 2 in line 30 to do this.

40 POKE 559,0

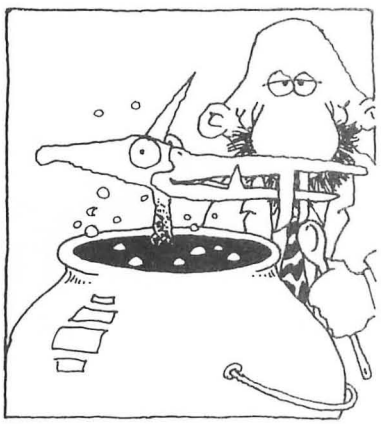
POKEing 559 with 0 disables the ANTIC microprocessor, speeding up the computer's calulating time by up to 30 percent. The screen will go blank while the computer is calculating. Don't panic! We'll turn the screen back on later in the program.

50 FOR X = 0 TO 511
60 POKE START + X,PEEK(57344 + X)
70 NEXT X

This is the sneaky part of the program. Line 60 transfers a copy of the character information, stored in ROM locations 57344 + X, to the area we've set aside, RAM locations START + X.

Now that we have our cast of characters in RAM where we want them, we can reshape and modify them—and really mess them up!

80 FOR LOCATION = 0 TO CHARACTERS – 1
90 FOR BYTE = 0 TO 7
100 READ SHAPE
110 POKE START + (LOCATION)*8 + BYTE + 264,SHAPE
120 NEXT BYTE
130 NEXT LOCATION



Here's where the actual character redefinition occurs. (Bubble, bubble, toil and trouble.) Since this is the heart of this great program, let's work it through step by step:

80 FOR LOCATION = 0 TO CHARACTERS – 1

This line assigns a LOCATION to each character. We're only redefining one character in this program, so it will be assigned to LOCATION 0. If we were redefining more than one character, the next character would be assigned to LOCATION 1.

I've subracted 1 from CHARACTERS (CHARACTERS – 1) because computers don't count like we do. Your Atari thinks that LOCATION 0 is the FIRST location, LOCATION 1 is the SECOND location, etc. Who says that computers are smart? When they count to 4, they start with 0. So, in a FOR/NEXT loop like FOR X = 0 TO 4, the computer will count FIVE cycles—cheeez!

90 FOR BYTE = 0 TO 7

Now we make room for each 8-byte character.

100 READ SHAPE

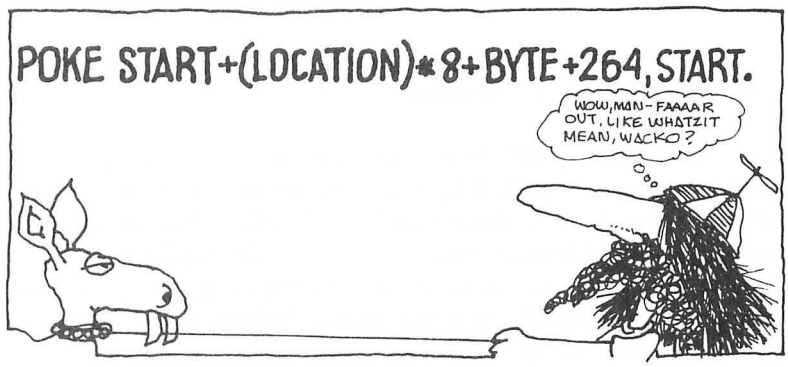
Line 100 reads the DATA "SHAPE" of each character and places it in the variable SHAPE in line 110.

110 POKE START+ (LOCATION)*8 + BYTE + 264,SHAPE

Line 110 is the BIGGIE! Follow closely as I get tongue-tied trying to work my way out of this one.

Here goes—the world's most complicated sentence:

In this stupendous line we POKE memory location START + (LOCATION) * 8 + BYTE + 264 with SHAPE.



OK then, let's try the world's longest sentence:

What we're doing here, in plain, but lengthy English, is POKEing the alternate character set, START, plus room for each character, (LOCATION) * 8, plus the 8 bytes that make up each character, plus the offset of letter you'd like to begin with, *with the SHAPE of the character you're redefining*.

Think about it!

You don't really have to understand why this line works to program arcade games! Just use it as shown. (I feel better already.)

In this example I've used 264 as the offset to redefine the letter *A*. If you'd like make your Wackenstein out of the letter *B*, for example, just change the offset to 272.

140 POKE 559,34

Our speedy little computer has completed its calculations. Now, POKEing 559 with 34 turns the ANTIC processor (and the screen) back on.

150 GRAPHICS 2:POKE 756,START/256

Line 150 first turns on graphics mode 2. Then it POKEs 756 with the location we've set aside (START/256). This clues in your computer about the redefined character set. From now on, your computer ignores its standard character set and goes directly to *your* cast of characters. You've got it eating out of your hands!

160 LOCATION = 0 TO CHARACTERS - 1
170 PRINT#6;CHR$(LOCATION + 65);
180 NEXT LOCATION

The moment of truth has finally arrived as we PRINT our newly redefined character on the screen.

The statement CHR$(LOCATION + 65); is loaded with subtleties. Since I decided to begin the redefined character set with the letter *A*, and since LOCATION = 0, LOCATION + 65 equals 65—*the ATASCII code of the letter* A! Wackenstein appears on the screen when we print the letter *A*. If you decide to start with the letter *B*, as I mentioned before, you'll have to replace 65, with 66, *B*'s ATASCII code.

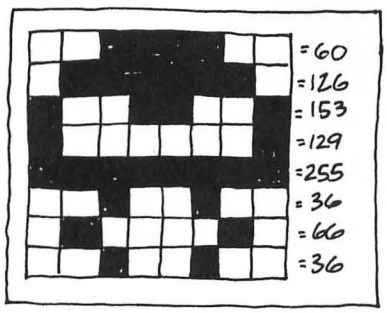
The semicolon at the end of line 170 serves a very important purpose. When we add more characters, they'll be displayed next to each other across the screen.

Have Some Fun!

Now it's time for a little experimentation. Here are some ideas:

1. Add POKE 708,99 to Line 170. (170 POKE 708,99:PRINT#6; CHR$(LOCATION + 65);) Wackenstein's color changes to pugnactious purple!
2. Redefine a different character by modifying the offset number in line 110 and its ATASCII code in line 170.

Steinenwack



Now that you understand how a character is redefined, it's time to prepare to move into the flip-flop world of animation.

To help you get set, I've drawn Wackentein's alter ego, Steinenwack:

Steinenwack has his legs closed, and his eyes are glazed! Just half a monster. But later, when he alternates places with Wackenstein, they'll come to life. One horrible, eeyukeee MONSTER!

I'll show you what Steinenwack looks like on your computer's screen. It's easy to do!

First change line 20 to read: 20 CHARACTERS = 2.

Now add this line of data to the Building Block program:

1010 DATA 60,126,153,129,255,36,66,36

That's all there is to it! RUN the program, then run out of the room before he comes *alive*!

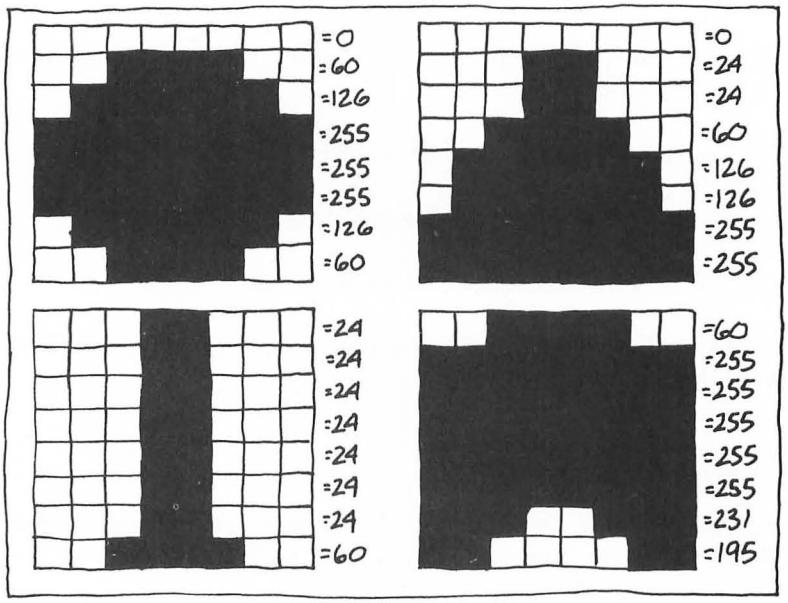
Multicolored Playfields

As horrible and disgusting as Wackenstein and his alter ego Steinenwack are, they still enjoy romping about on a colorful, interesting, and informative playfield.

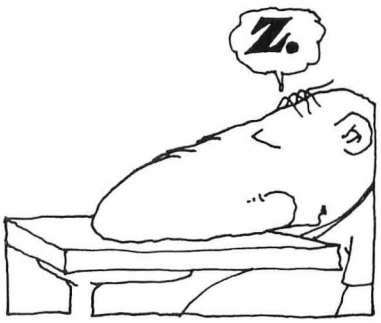
Your characters should have a nice place to play in, too! You already know how to design a playfield using COLOR, PLOT, and DRAWTO statements. Now, with Captain Action's help, I'll show you how to liven up your screen with *five* brilliant colors and contorted shapes, all using character graphics!

The Captain spent hours drawing these four simple shapes.

The numbers listed to the right of each shape are used in the DATA statements of this modified Building Block #1 program.



Enter this *brilliant* program, RUN it, then return to the classroom and I'll entertain you with one of my shorter lectures.



Colorful Playfield

```
10 . BUILDING BLOCK #1 MODIFIED-PLAYFIELD
   DESIGN
12 .
20 CHARACTERS = 4
22 .
30 START = (PEEK(742) - 4) * 256
40 POKE 559,0
42 .
50 FOR X = 0 TO 1023
60 POKE START + X,PEEK(57344 + X)
```

```
 70 NEXT X
 72 .
 80 FOR LOCATION = 0 TO CHARACTERS - 1
 90 FOR BYTE = 0 TO 7
100 READ SHAPE
110 POKE LOCATION*8 + BYTE + START +
    264,SHAPE:. *** Start at ATASCII 65 - The
    letter 'A'***
120 NEXT BYTE
130 NEXT LOCATION
132 .
140 POKE 559,34
150 GRAPHICS 2:POKE 708,76:POKE 712,99:POKE
    756,START/256
152 .
160 COLOR 65:PLOT 9,3: .*Standard upper case letter.
    POKE 708 to change character's color.*
170 COLOR 66 + 32:PLOT 9,4: .*Standard lower case
    letter. POKE 709 to change character's color.*
180 COLOR 67 + 128:PLOT 9,5: .*Inverse upper case
    letter. POKE 710 to change character's color.*
190 COLOR 68 + 160:PLOT 9,6: .*Inverse lower case
    letter. POKE 711 to change character's color.*
192 .
200 COLOR 65:PLOT 0,0:DRAWTO 19,0:DRAWTO
    19,9:DRAWTO 0,9:DRAWTO 0,1
202 .
1000 DATA 0, 60, 126, 255, 255, 255, 126, 60
1010 DATA 0, 24, 24, 60, 126, 126, 255, 255
1020 DATA 24, 24, 24, 24, 24, 24, 24, 60
1030 DATA 60, 255, 255, 255, 255, 255, 231, 195
```

BRILLIANT, isn't it? Now, that's my idea of a spiffy screen.

The programming to accomplish this spectacular result is simple and straightforward.

After adding the DATA statements, lines 1000 through 1030, I changed line 20 to read: CHARACTERS = 4; 4 characters are redefined!

Line 110: I began my character set with the letter *A* by inserting its offset number, 264, in line 110.

Line 150: Next I POKEd the standard uppercase color register with a nifty color (POKE 708,76) and painted the background color purple with a simple POKE 712,99.

Lines 160 TO 190: Here's where the PLOT thickens. I PLOTed each of the four characters one on top of the other and centered them on the screen.

The four characters are actually the redefined standard uppercase letters *A*, *B*, *C*, and *D*. By adding 32, 128, and 160 to *B*, *C* and *D* respectively, Ive changed them to standard lowercase, inverse uppercase, and inverse lowercase letters respectfully—Ahem.

Mess with the Color

When the word READY appears in the blue text window, it's your turn to mess up each color by POKEing its assigned color register. Try entering my favorite, POKE 709,99 <RETURN>. Pugnactious purple! When you get sick of that one (I am), be adventurous and experiment with other color registers and combinations. Bet you can't do worse than I did!

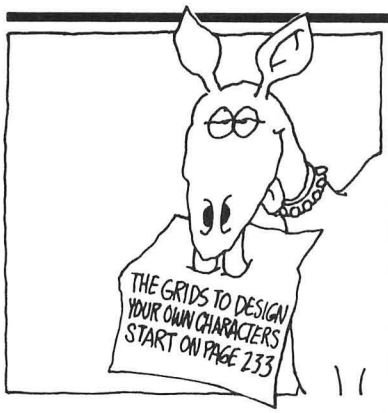
Line 200: I've used the redefined COLOR 65 in line 200 to PLOT, then DRAW the border around the screen.

Not bad, huh?

In addition to changing each character's color, you can reposition any of the characters by changing the numbers that follow its PLOT statement. And, of course, you can PLOT more of them on the screen!

Your Foundation to Better Playfields

Use this simple and straightforward example as your foundation as you design your game's playfield. Make JUMBO letters for use in custom readouts. Draw complex scenery. Draw Albert Einstein. Draw pronouns! It's all possible now that you know how simple it is to do.



Design Your Playfield on Paper First

The best way to design your screen playfield is with a piece of gridded paper sized to the graphics mode you'll be using. First draw your playfield to scale, then transfer your creation to the screen.

Mix Techniques

The Atari character set includes many predefined graphics characters. For instance, decimal code 20 is a circle. If you haven't redefined these graphic characters, they are still standing by awaiting your command! Using these graphic characters with the special characters you've designed will add pizzazz to your playfield!

Time Well Spent

Spend time with this short program. Really get to know it well, then use it to create the best playfields in the universe! Like Captain Action did. . . ?

The Monster Maker

Are you tired of using pencil and paper to create your monsters and playfields? Fed up with having to write *flawless* DATA statements? Nobody's perfect; even I make *misstakes*!

Do you want instant monsters? Do you go into orbit playing with a joystick? Would you like to mold your monsters on a screen rather than on a piece of paper?

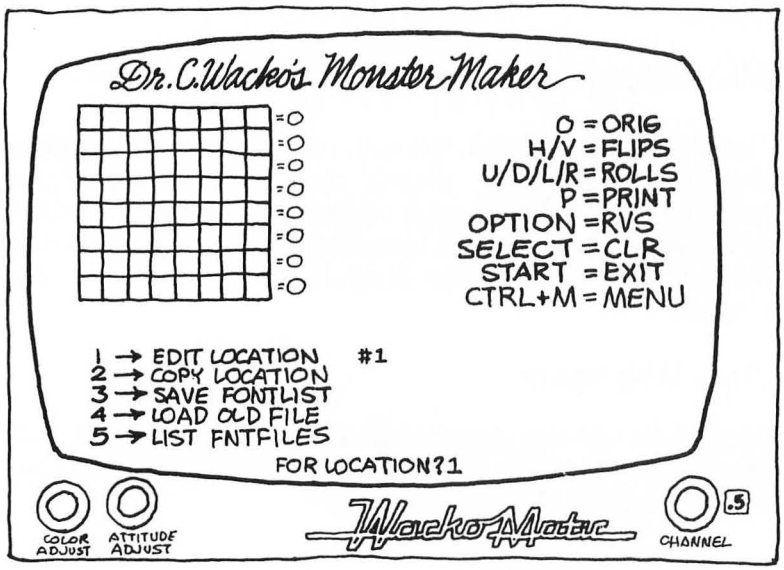
Well, the answer to your wildest fantasies is here! Dr. C. Wacko's *FABULOUS* (and a little bit spectacular) Monster Maker!

The Monster Maker program is on page 201. It's lo-o-ng. But don't let that deter you. Be determined, take your time, enter this program, and SAVE it. The Monster Maker will turn you into a certified arcade-game-designing wacko!

I'm going on vacation for a few weeks while you type and enter this program.

Now that I'm tanned and relaxed, I'll explain all of Monster Maker's superdeluxe features.

But first RUN Monster Maker so we can go through its operation together.



Neat screen, huh?

Lots of stuff is presented on the screen. That's because Monster Maker can do lots of real neat stuff!

Let's start with. . . .

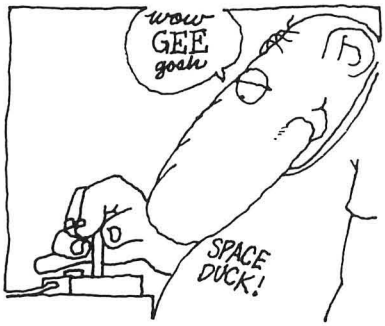
1. EDIT LOCATION

Just press the number 1 <RETURN> on your computer's console and see what happens. In typical computer fashion it asks a question: FOR LOCATION?

You might ask, "How am I supposed to know what to enter in response to FOR LOCATION? I'm supposed to tell you!

Up to 64 Monsters

The Monster Maker can help you design up to sixty-four monsters, gargoyles, or munchkins, or. . . . But if your goal is to bring your fantasies to animated life it's best to draw your first character in LOCATION 1. So, PRESS 1 <RETURN>.



A Winking, Blinking Cursor!

That's what you'll see on your electronic piece of paper! Plug your joystick into port 1, and start moving the cursor around.

"Press De Button, Mon"

Having fun? Just press the joystick's red button to draw your picture. When the button is depressed (no pun intended), the cursor draws over blank areas and erases areas that have been drawn on.

Draw Your First Monster

Using your creative talent (and lots of luck), draw a replica of Wackenstein.

Finished?

B.E.A.U.T.I.F.U.L. Looks GREAT!

Do you recognize those numbers to the right of Wackenstein? Right! They're the numbers that make up the DATA statement in Building Block #1, page 54.

You might ask, again: "But, what do I do now?"

Press START to Store It!

Now that you've finished your first creation, just press START to store Wackenstein in LOCATION 1.

Now you can move on to LOCATION 2 and create Steinenwack!

2. COPY LOCATION

An easy way to draw Steinenwack is to copy Wackenstein into LOCATION 2, then modify him to suit your needs.

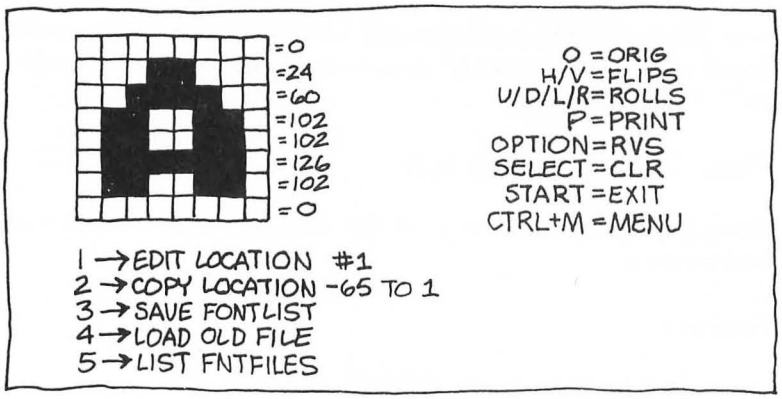
Let's go for it.

Press 2 <RETURN>. You'll be asked to supply two numbers, FROM and TO, separated by a comma. To transfer an image of Wackenstein *from* LOCATION 1 to LOCATION 2, just enter: 1,2 <RETURN>.

AMAZING! Wackenstein's twin appears in LOCATION 2.

Now just perform a little plastic surgery on Wackenstein, mold him into Steinenwack, and press START to store him in LOCATION 2.

Redefine Any Character



The COPY LOCATION function also lets you redefine *any* letter or character in your Atari's character set. Here's how this wondrous feature works.

After pressing 2 <RETURN>, enter the ATASCII code of the letter or symbol you'd like to see displayed, preceded by a minus sign (−). Then enter a comma and the LOCATION number.

For example: If you'd like to redefine the capital letter *A* in LOCATION 1, enter:

−65,1 <RETURN>

The letter *A* will appear ready for modification. Refer to the ATASCII chart on page 189, and use this great feature to see how your Atari makes its characters.

3. SAVE FONTLIST

You've just created a FONTLIST! Wakenstein and Steinenwack are your FONTLIST. They are redefined characters. Now it's time to SAVE them so you can use them later in a game or watch them in action when I show you my next creation: Wacko's Animation Tester.

So, let's SAVE'em. First—and this is very IMPORTANT!—if you're working on disk, make sure you've got a formatted disk with DOS and lots of free sectors in your disk drive. Wheew! almost SAVEd your first font to NOTHING!

SAVE IT?

After you press 3 <RETURN> you'll be asked: SAVE IT? Just reply with a resounding YES <RETURN>. (If you had pressed the "3" key by mistake, you'd reply by typing in: NO <RETURN>.)

How many CHARACTERS?

Next, the question CHARACTERS? will appear, asking how many characters you want to SAVE. Since you've only created two monsters, type in 2 <RETURN>.

D:NAME?

What do you want to call your new font? Make up a name. Just make sure it's not more than *eight* characters long. I used MONSTER, then answered this prompt by typing:

D:MONSTER <RETURN>

That's all there is to it! Your first redefined character set is now safely SAVED on your disk.

Attention Cassette Owners

If you're using a cassette just type:

C:MONSTER <RETURN>

4. LOAD OLD FILE

Now that you've got your first font neatly saved on your disk or cassette, it's time to show you how to load it back into Monster Maker.

Loading a font into Monster Maker is easy. Press 4 <RETURN> and answer the prompts. The loading sequence is identical to the saving sequence. Just follow the bouncing prompts!

See Your Newly Loaded Font

Go to the EDIT LOCATION mode, by pressing 1 <RETURN>, to see your newly loaded font.

5. LIST FNTFILES

If you want to see what fonts are stored on your disk, press 5 <RETURN>. All their names will be displayed on your screen. AMAZING! To exit this mode, press START.

Other Options and Neat Things

All of Monster Maker's special editing features are displayed on the right side of the screen. Here's how they work and what each one does:

- O = ORIG: Press O to restore a shape you've been working on to its original configuration.
- H/V = FLIPS: Press H to flip your character left to right or right to left. This feature is real handy when you're designing a character that must face left in one animation sequence and right in another. Press V to flip a character upside down!
- U/D/L/R = ROLLS: Press U to scroll your character up, D to scroll down, L to scroll left, and R to scroll right.
- P = PRINT: Press P to print your character on your Atari printer, complete with data numbers and LOCATION number!
- OPTION = RVS: Press the OPTION key to reverse the image of a character. Press OPTION again to restore it.
- SELECT = CLR: Press the SELECT key to clear the image from the work area.
- START = EXIT: Press the START key when you've finished editing each character. The character will then be stored in the LOCATION you've chosen.
- CTRL + M = MENU: This option is only included in the book/disk package of Dr. C. Wacko. Pressing the CTRL key plus the M key returns you to the disk's main menu.

Put Your Font into Your Program

You've created MONSTER.FNT and saved it to your disk or cassette. Now I'll show you how easy it is to see it displayed on your screen.

Modify the Building Block "1 program. For all you disk drive owners, add these three new lines to Building Block #1:

```
 72 CLOSE #1:OPEN#1,4,0,"D:MONSTER.FNT"
 74 GET #1, CHARACTERS
135 CLOSE #1
```

Next, change line 100 to read:

```
100 GET #1,SHAPE
```

That's all you have to do! Ensure that MONSTER.FNT is stored on the disk that's in the drive, then RUN your modified program.

Eeeyuk! There they are again!

If you're using a cassette, simply modify line 72 like this:

```
72 CLOSE #1:OPEN #1,4,0,"C:"
```

Everything else remains the same, including the "Eeeyuk!"

Using Another Font

To see one of your other creations displayed, just change the name of the of the font following D: in Line 72 to match the name of any font that you've saved on disk.

Fool'n Around

Now's a good time to pause, reflect, relax, and go absolutely *wild*! (Not necessarily in that order.)

Practice with Monster Maker. Make wild and crazy characters, load them into your modified Building Block #1 program, and watch them appear on your screen!

Try different graphics modes! What will your creatures look like in graphics modes 0 or 1?

REMEMBER: If you use graphics mode 0 you'll have to:

- Change Line 30 to read: START = (PEEK(742) – 4) * 256
- Change Line 50 to read: FOR X = 0 TO 1023
- Change Line 150 to read: GRAPHICS 0:POKE 710,0:POKE 756,START/256

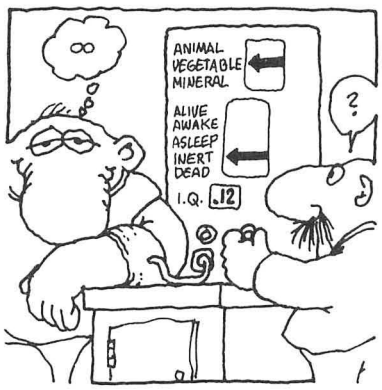
Let's Get MOVING!

Wackenstein's a pretty neat-looking monster. But he's just not going anywhere!

Turn the page, if you dare, and we'll bring Wackenstein to LIFE!. Uh-Oh!

4
Flip-Flop Animation

The time has finally arrived. Wackenstein's coming to life!



Presenting: Wacko's Animation Tester

The Animation Tester, on page 71, represents years of research, drinking margaritas, and suffering subsequent hangovers. So when you enter this program, puleeeze, do it quietly. Shhhhhhh!

E-Z Operating Instructions

1. RUN the program! If you're scared, call in a large friend, or simply run away.

2. To see what fonts are stored on your disk, enter Y. If you don't care, or already know, enter N.

3. Indicate the font of your choice like this:

D:MONSTER <RETURN> (Disk)

C: <RETURN> (Cassette)

You don't have to add the extension—.FNT. The program does it for you!

4. Select the graphics mode in which you'd like to see Wackenstein (or any other font) appear, by entering either 0, 1, or 2 and pressing RETURN.

5. Enter the animation speed and press RETURN. 5 is extremely fast and 500 is v . . . e . . . r . . . y s . . . l . . . o . . . w. 100 is good for starters.

Wackenstein's Alive!

6. Press START to try another font or change the graphics mode or speed settings.

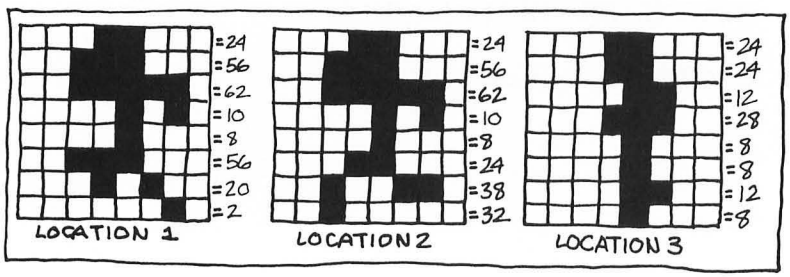
It's neat to watch the fonts you've created flip-flop on your screen! The Animation Maker is my gift to game designers: It's a great tool that can help you review the animated characters you've designed for your arcade games.

Now it's time to show you how to add basic animation to your games.

So far, you've learned two ways to enter data into a character redefinition program.

METHOD 1: Enter data from data statements in your program.
METHOD 2: Enter data from a font file that has been stored on a disk or cassette.

In the example I'm about to show you, I've used Method 1. You can alter this program to read characters from a font file by making the simple modifications I showed you on page 69.

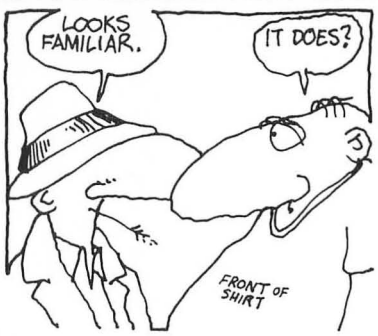


Here's a new set of three characters. When we flip through them, the man will appear to be jogging. Actually he looks like he's on a treadmill running to nowhere. (Sounds like someone I know. Sigh! Woe is me! It isn't always easy being a wacko.)

The data that make up each character are listed in lines 1000 through 1020 of this program:

Flip-Flop Jogger

```
 10 REM BUILDING BLOCK #1: CHARACTER
    REDEFINITION WITH FLIP-FLOP ANIMATION
    FROM DATA
 20 CHARACTERS = 3
 30 START = (PEEK(742) - 2)*256
 40 POKE 559,0
 50 FOR X = 0 TO 511
 60 POKE START + X,PEEK(57344 + X)
 70 NEXT X
 80 FOR LOCATION = 0 TO CHARACTERS - 1
 90 FOR BYTE = 0 TO 7
100 READ SHAPE
110 POKE (LOCATION)*8 + BYTE + START + 264,
    SHAPE
120 NEXT BYTE
130 NEXT LOCATION
140 POKE 559,34
150 GRAPHICS 2:POKE 756,START/256
152 .
154 .
160 FOR LOCATION = 0 TO CHARACTERS - 1
170 COLOR (LOCATION + 65)
180 PLOT 9,5
190 FOR PAUSE = 1 TO 100:NEXT PAUSE
200 NEXT LOCATION
210 GOTO 160
220 .
230 .
1000 DATA 24, 56, 62, 10, 8, 56, 20, 2
1010 DATA 24, 56, 62, 10, 8, 24, 38, 32
1020 DATA 24, 24, 12, 28, 8, 8, 12, 8
```



You're right! It's our old friend, Building Block #1 Only lines 170 through 210 (the lines used to display the characters) have been changed. Whoops, I almost forgot! Since there are now three characters, the value of CHARACTERS in line 20 is now 3 (CHARACTERS = 3).

Lines 160 through 210 Explained

The first in this set of three characters is a redefined letter A (ATASCII code 65).

During the FOR/NEXT loop's first cycle (line 160), the value of LOCATION equals 0. So in line 170, COLOR (0 + 65) = 65, A's ATASCII value! COLOR 65 prints our first frame at coordinates 9,5.

During the next cycle, LOCATION will equal 1 and the redefined letter *B* (COLOR 66) will appear on your screen

The last character to appear will be the redefined letter *C* (COLOR 67). Then this short routine goes to line 210, which loops back to line 160 to begin the process over again.

That's all there is to it—simple flip-flop animation. The principle is the same as for a movie. We just print a series of "frames" on top of each other, one after the other.

Machine-Language Flip-Flop Jogger

Many years ago, when I was still in nursery school, I invented the Machine-Language Machine. I thought it would help me make my arcade games more efficent by creating hundreds of machine-language routines. How wrong I was. It has only developed two

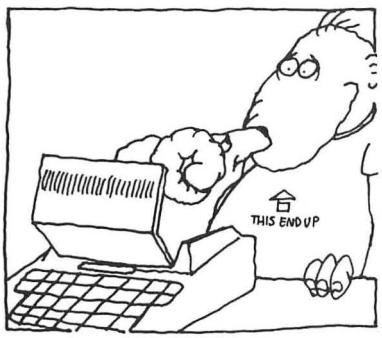
routines in all these years. One routine for MOVEMENT and another for JOYSTICK CONTROL! It did learn how to speak in Sanscript and recite the Kama Sutra. . . not a complete loss.

Using the USR Function

You can use USR to load a machine-language routine into your games. When your program bumps into a USR function, it goes berserk! It stops everything and starts looking for a machine-language program to execute! (Off with his head!)

But first you've got to put a machine-language routine into a specific memory location in your computer and then remember to specify it so the USR function can find it. After all, it's not nice to frustrate a USR.

Those nice people at Atari put aside an area in your computer that can be used to store machine-language routines and other stuff. It's sometimes referred to as "Page 6" because it's the sixth set of locations in RAM (trivia).



Put It in Locations 1536 to 1791

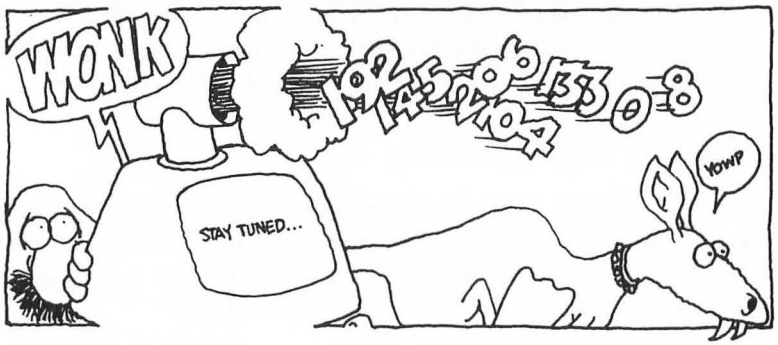
Page 6, also called *Free RAM*, begins at LOCATION 1536 and ends at LOCATION 1791. Put your machine-language routines, and other stuff, in these locations so they won't get in the way of the other things that your Atari's trying to do, like RUN your programs and drink soda.

Now that you know where to put a machine-language routine I'll show you my Movement Routine, how to put it into Page 6, and (amazingly enough) how it's used to animate the Jogger.

So, without further ado, here's the machine-language routine that the Machine-Language Machine developed for me!

104, 104, 133, 204, 104, 133, 203, 104, 133, 207, 104, 133, 206, 160, 0, 177, 206, 145, 203, 200, 192, 8, 208, 247, 96

Disappointed? I was, when the Machine-Language Machine first spit these numbers out.



But these numbers have a mystical quality. When put into the computer's memory, then recalled and used by a USR function, they let you perform some very magical animation!

Some Magical Animation

By using this Machine-Language Movement Routine you'll be able animate the Jogger, just like you did in the Flip-Flop Jogger program. *But you'll only be redefining and animating one character, not three!*

What's He Talking About?

In the Flip-Flop Jogger program we redefined and animated the letter *A*, then the letter *B*, and finally the letter *C*. By using machine-language movement you'll be able to create the same effect. And you'll only have to redefine and animate one character— the letter *A*, for example.

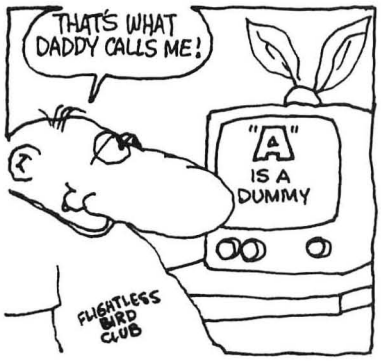
You'll also be able to animate parts of the character. Just the top of the letter *A*, if you want. You can use this feature to move the eyes of a character, or make it stick out its tongue!

Using a USR

Coming up, down below (huh?), is a short program that uses a USR function with our Machine-Language Movement Routine.

Here's what our USR function looks like:

A = USR(SL,START + 264,SL1 + FR*8)



I'll break this USR function into itsy-bitsy elements to show you what each element does. Here's how it works:

A = USR

The letter *A* is called a *dummy*. You can use any letter you like. (Just use something!) Its function is to activate the USR function.

SL

SL represents the *S*tarting *L*ocation (in the computer's memory) of the machine language routine. If you place the routine in Page 6, SL will equal LOCATION 1536.

START + 264

START + 264 is the location (in the computer's memory) of the first character you'd like to redefine and animate. In this example we're going to redefine and animate the letter *A*—offset 264. If you'd like to use another letter, just change the offset number.

SL1

SL1 represents the starting location of the data used to redefine the character.

FR*8

FR*8 is added to SL1 to call up each eight-byte frame for animation in this program.

Putting It All Together

This USR function and machine-language routine cause the character you've selected, *A*, to be continuously redefined with the Jogger's (or any other) data.

Type in and RUN this program, then I'll explain how it all fits together.

Machine-Language Flip-Flop Jogger

```
 10 . MACHINE LANGUAGE MOVEMENT WITH DATA
    STATEMENT LOADING
 12 .
 14 .
 20 START = (PEEK(742) - 2) *256
 30 POKE 559,0
 40 FOR X = 0 TO 511
 50 POKE START + X,PEEK(57344 + X)
 60 NEXT X
 62 .
 64 .
 66 . PUT MACHINE LANGUAGE ROUTINE INTO
 PAGE  68
 70 FOR A = 1536 TO 1560
 80 READ B
 90 POKE A,B:NEXT A
100 SL = 1536
102 .
104 .
106 . PUT JOGGER'S DATA INTO FREE RAM
110 FOR A = 1561 TO 1584
120 READ B
130 POKE A,B:NEXT A
140 SL1 = 1561
142 .
144 .
150 POKE 559,34
160 GRAPHICS 2:POKE 756,START/256
162 .
164 .
166 . ANIMATION ROUTINE
170 FR = 0
180 FR = FR + 1:IF FR>2 THEN FR = 0
190 FOR PAUSE = 0 TO 25:NEXT PAUSE
200 A = USR(SL,START + 264,SL1 + FR *8)
210 COLOR 65
220 PLOT 9,5
230 GOTO 180
232 .
234 .
240 . MACHINE LANGUAGE MOVEMENT DATA
```

```
250 DATA 104, 104, 133, 204, 104, 133, 203, 104, 133,
    207, 104, 133, 206, 160, 0, 177, 206, 145, 203, 200,
    192, 8, 208, 247, 96
252 .
254 .
260 . CHARACTER'S DATA (JOGGER)
270 DATA 24, 56, 62, 10, 8, 56, 20, 2
280 DATA 24, 56, 62, 10, 8, 24, 38, 32
290 DATA 24, 24, 12, 28, 8, 8, 12, 8
```

Lines 20 through 60: You know how these work already!

Lines 70 through 90: These lines READ the machine language movement data numbers (line 250) and place them in locations 1536 to 1560.

Line 100: This line assigns the variable SL1 (the starting location of character data) the value 1561.

Lines 110 through 140: Place the character's data into Free RAM locations 1561 to 1584.

Lines 150 and 160: These lines turn the screen back on, turn on graphics mode 2, and activate the alternate character set.

Line 170: Sets FRAME counter to zero.

Line 180: Is designed to display *three* characters before resetting to zero. FR has values of 0, 1, and 2, then resets to zero. The Jogger is made of *three* shapes. The value used in the IF statement will change depending on the number of shapes you're going to animate. If you're going to animate four shapes, for example, just change it to read FR>3.

Line 190: This FOR/NEXT loop slows down the action so you can see the animation. Remove this line to see how fast the Jogger can run!

Line 200: This is where all the redefinition/animation takes place. The frame counter (FR*8) points to the location of the data for each eight-byte character one after another. The USR statement then replaces the letter *A* with one eight-byte character each time the Animation Routine cycles.

Lines 210 and 220: These lines display the letter *A*'s redefined shapes at coordinates 9,5.

Line 230: Returns to 180, where the next frame is selected and the whole process begins again.

An Important Number in Line 250!

Line 250 contains the Machine-Language Movement Routine. The number 8 (fourth from the end) is extremely IMPORTANT!.

Because the Jogger program only redefines and animates one 8-byte letter(the letter *A*) the number 8 is used in the machine-language routine. If you'd like to see two joggers run nowhere on your screen, add these two new lines and change 8 to 16:

222 COLOR 66
224 PLOT 9,6

Each time you want to animate another character, add 8 to this number!

Animate Half a Jogger!

To animate only a portion of the letter *A* replace the number 8 with any number between 1 and 7. If you'd like to see what animating just the top half of the letter *A* looks like, just replace the 8 with 4. Now, in an arcade game, you'll be able to make any portion of the character move independently of the rest of its body! Experiment with this concept. Put on your thinking cap and try plotting characters on top of one another, next to each other, or in the dishwasher, while changing the value of this special number.

Weird Harold

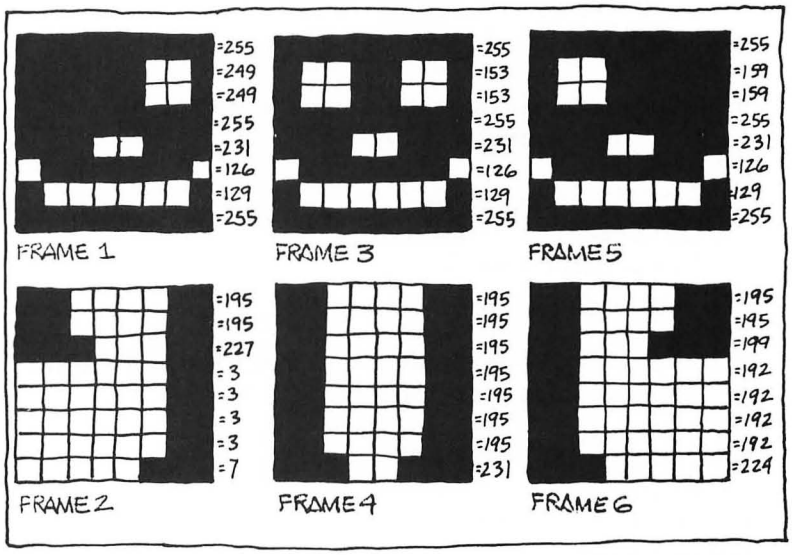
Junior's been coming home with some pretty strange friends since he became a teenager. But the strangest of all is a gawky fourteen-year-old called Weird Harold.

After meeting this young man, I was driven to bring him to life on the computer screen and share his special weirdness with you.

Trying to replicate weird on a computer screen is no easy task. It was difficult simulating Harold's effervescent fragrance. What I now present, in all its prancing glory, is a mere caricature of this strange fellow.

But it is weird! Trust me!

I first designed Harold using the Monster Maker. No mean *feet*. Here's what I came up with:



Frame 1 is Harold's head—one eye is closed. Frame 2 shows Harold's feet. He's dancin' and prancin' just as I remember him. Frames 3 through 6 follow this very *important* sequence. We'll use the numbers to the right of each frame, *in the order shown*, in the Amazing Feet program!

My ingenious plan was to flip-flop animate *two frames* at a time. One positioned on top of the other! Junior thought I was getting a little bit carried away. But, as you'll see, the actual programming is very similar to the Flip-Flop Jogger program we just played with. No problem!

Don't worry! Here's the program. Look it over, enter it, RUN it, then open the windows. Some of Harold's fragrance may slip out.

Amazing Feet

```
  5 . Flip-Flop WEIRD HAROLD
 10 . MACHINE LANGUAGE MOVEMENT WITH DATA
    STATEMENT LOADING
 12 .
 20 START = (PEEK(742) - 2)*256
 30 POKE 559,0
 40 FOR X = 0 TO 511
 50 POKE START + X,PEEK(57344 + X)
 60 NEXT X
 62 .
 64 . Put Machine-Language routine into Page 6 ..
    Starting Location = 1536
 66 .
 70 FOR A = 1536 TO 1560
 80 READ B
 90 POKE A,B:NEXT A
100 SL = 1536
102 .
104 . Put Weird Harold's Data into free RAM .. his Data
    Starts at Location = 1561
106 .
110 FOR A = 1561 TO 1608
120 READ B
130 POKE A,B:NEXT A
140 SL1 = 1561
142 .
150 POKE 559,34
160 GRAPHICS 2:POKE 756,START/256
162 .
164 . Animation Routine - Note the number 16 in USR
    routine!
166 .
170 FR = 0
180 FR = FR + 1:IF FR>2 THEN FR = 0
190 FOR PAUSE = 0 TO 25:NEXT PAUSE
200 A = USR(SL,START + 264,SL1 + FR*16)
202 .
204 . PLOT Characters - one on top of the other
206 .
210 COLOR 65:PLOT 9,5
220 COLOR 66:PLOT 9,6
222 .
230 GOTO 180
```

```
 232 .
 240 . Machine-Language Movement Data
 242 .
 250 DATA 104, 104, 133, 204, 104, 133, 203, 104, 133,
     207, 104, 133, 206, 160, 0, 177, 206, 145, 203, 200,
     192, 16, 208, 247, 96
 252.
 260 . Weird Harold's Data
 262 .
1000 DATA 255, 249, 249, 255, 231, 126, 129, 255
1010 DATA 195, 195, 227, 3, 3, 3, 3, 7
1020 DATA 255, 153, 153, 255, 231, 126, 129, 255
1030 DATA 195, 195, 195, 195, 195, 195, 195, 231
1040 DATA 255, 159, 159, 255, 231, 126, 129, 255
1050 DATA 195, 195, 199, 192, 192, 192, 192, 224
```

Quite a guy, isn't he? Weird Harold may be strange, but this program should be very familiar to you. It's almost identical to the Machine-Language Flip-Flop Jogger on Page 78.

Only a few things make Weird Harold unique (besides his fragrance).

The FRAME Counter Is a Bit Unique

Take a look at lines 180 through 200. In line 180 the frame counter counts from 0 to 3 because Harold's two parts (head and dancin' feet) are displayed at the same time. Each set counts as one frame.

The USR Statement Is a Trifle Unique

Harold's two parts (two lines of DATA, 16 bytes) are flipped into the USR routine as a set. That's why the last phrase of the USR statement is FR*16 and not FR*8 as it was in the Jogger program.

Two Characters Are PLOTed on Top of Each Other?

Line 210 PLOTs a redefined letter *A*, Harold's head. Line 220 PLOTs the redefined letter *B*, his amazing feet, exactly where they belong, below his head! If you didn't enter the data numbers in the order shown in the frame drawings, things might have gotten pretty confusing. Try messing up the order of the DATA lines. You'll get some very strange results.

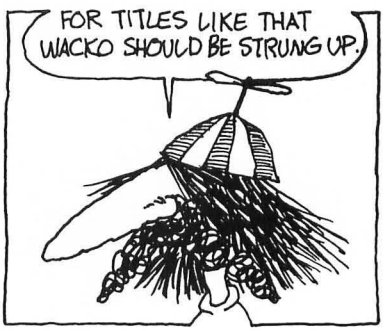
But the Machine-Language Movement Routine Is Really Weird!

One last detail. Remember that real important number in the Machine-Language Movement routine? The fourth from the end. Check it out! I've changed that number to 16 because the USR routine is redefining *two* characters, the letters *A* and *B*.

Go with Gusto!

Don't limit yourself to animating two-part weirdos! Rush to your friendly Monster Maker and design multi-multipart weirdos! Just make the appropriate changes in the unique parts of the program and you're on your way to weirdodom!

Using Strings to Enter DATA, Or, Don't String Me Along



Using strings to enter data shortens your programs, saves data loading time, and lets you forget about searching for free memory loctions to store it all in!

CAUTION: Read carefully! Using strings is very easy to do, but very hard to explain. No matter what happens, just do what I do, not what I say. (What?)

Repeat after me: "Simple Simon says, Strings save serendipitous salamanders."

I warned you!

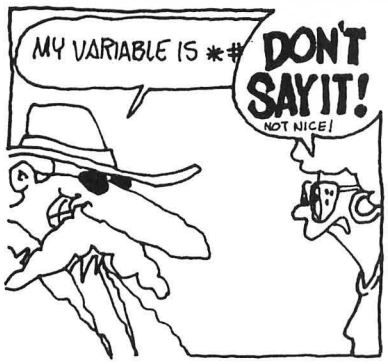
Here's How to Do It

Convert the data you're going to use in your program into ATASCII code, then assign the whole mess to a string variable.

Look at the ATASCII code chart on page 189 while I show you what I'm talking about. Suppose your data contained these three numbers: 104, 75, and 47. Your string variable would look like this:

A$ = "hK/"

Just consider each data number as a decimal code and replace it with its associated ATASCII Character. I've named my string



variable A$. You can name yours anything you want . . . within the limits of decency!.

One Serious Limitation

If your data statement contains the numbers "34" or "155" you can't use string loading. Your computer won't accept the ATASCII codes for these two numbers, because 34 = " , which ends the string, and 155 = EOL (end of line), which does the same. They are the ATASCII codes for invalid string data.

Another, Not-So-Serious Limitation

All data must be in in the form of positive numbers ranging from 0 to 255.

Convert The Movement Routine to A String

I'll show you how to convert the Machine-Language Movement Routine to its string equivalent; then we'll use it in the Machine-Language Flip-Flop Jogger program instead of the DATA statement.

E.Z. MACHINE LANGUAGE MOVEMENT
TO STRING CONVERSION

VALUE	BYTE	KEYSTROKES
104	1	(LOWR) h
104	2	(LOWR) h
133	3	(RVS) CTRL-E
204	4	(RVS) L
104	5	(LOWR) h
133	6	(RVS) CTRL-E
203	7	(RVS) K
104	8	(LOWR) h
133	9	(RVS) CTRL-E
207	10	(RVS) 0
104	11	(LOWR h
133	12	(RVS) CTRL-E
206	16	(RVS) 1
206	17	(RVS) N
145	18	(RVS) CTRL-Q
203	19	(RVS) K
200	20	(RVS) H
192	21	(RVS) @
8	22	CTRL-H
208	23	(RVS) P
247	24	(RVS) n
96	25	CTRL-.

Now, A$ contains the Machine-Language Movement Routine converted into ATASCII characters.

Yes folks, here it is. The highly EDIFYING, DEATH-DEFYING, GRATIFYING AND, and AMAZING . . .

The "Don't String Me Along" Flip-Flop Jogger

```
10 . STRING LOADING OF MACHINE LANGUAGE
   MOVEMENT ROUTINE
12 .
14 .
20 CLR :DIM A$(25)
30 A$ = "MOVEMENT ROUTINE DATA CONVERTED TO
   ATASCII CHARACTERS"
40 SL = ADR(A$)
42 .
44 .
50 READ B
60 START = (PEEK(742) - 2)*256
70 POKE 559,0
80 FOR X = 0 TO 511
90 POKE START + X,PEEK(57344 + X)
100 NEXT X
102 .
104 .
110 RESTORE 260
120 FOR A = 1536 TO 1559
130 READ B
140 POKE A,B:NEXT A
150 SL1 = 1536
152 .
154 .
160 POKE 559,34
170 GRAPHICS 2:POKE 756,START/256
172 .
174 .
180 FR = 0
190 FR = FR + 1:IF FR>2 THEN FR = 0
200 FOR PAUSE = 0 TO 25:NEXT PAUSE
210 A = USR(SL,START + 264,SL1 + FR*8)
220 COLOR 65
230 PLOT 9,5
240 GOTO 190
```

```
250 .
252 .
260 DATA 24, 56, 62, 10, 8, 56, 20, 2
270 DATA 24, 56, 62, 10, 8, 24, 38, 32
280 DATA 24, 24, 12, 28, 8, 8, 12, 8
```

By using the string loading technique in lines 30 through 40, we've eliminated a line of data and the routine used to read it into memory location 1536. See how easy it is to use!

The Final Refinement: No DATA Statments!

The next program is really streamlined. It uses string loading to replace *both* sets of data! The Movement Routine and the characters. It's what's known in the biz as a superrefined grade A program.

Convert the Jogger's data to ATASCII characters, then assign them to string variable B$ in line 40. Don't forget that B$ must be DIMensioned to the number of bytes your character(s) contains. In this program B$ is DIMensioned to accept 24 bytes— DIM B$(24).

The All-String Grade A Flip-Flop Jogger

```
 10 REM MACHINE LANGUAGE MOVEMENT WITH
    STRING LOADING
 12 .
 14 .
 20 CLR :DIM A$(25),B$(24)
 30 A$ = "MOVEMENT ROUTINE DATA
    CONVERTED TO ATASCII CHARACTERS"
 40 B$ = "CHARACTER'S DATA CONVERTED TO
    ATASCII CHARACTERS"
 50 SL = ADR(A$):SL1 = ADR(B$)
 52 .
 54 .
 60 START = (PEEK(742) - 2)*256
 70 POKE 559,0
 80 FOR X = 0 TO 511
 90 POKE START + X,PEEK(57344 + X)
100 NEXT X
102 .
104 .
```

```
110 POKE 559,34
120 GRAPHICS 2:POKE 756,START/256
122 .
124 .
130 FR = 0
140 FR = FR + 1:IF FR>2 THEN FR = 0
150 FOR PAUSE = 0 TO 25:NEXT PAUSE
160 A = USR(SL,START + 264,SL1 + FR*8)
170 COLOR 65
180 PLOT 9,5
190 GOTO 140
```

Now you know three ways to enter data into your game programs:

1. Enter data from DATA statements.
2. Enter data from a font file that's been stored on disk or cassette.
3. Enter data using a string.

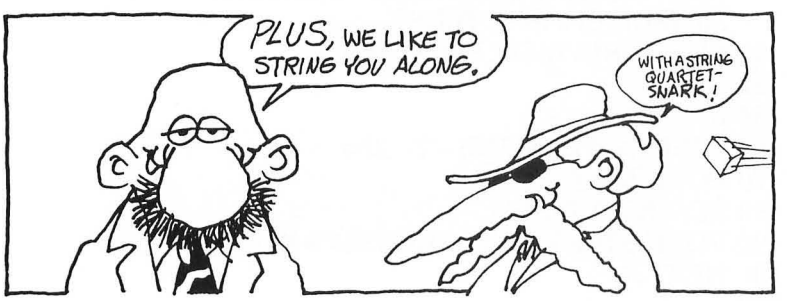
What's so great about string loading?

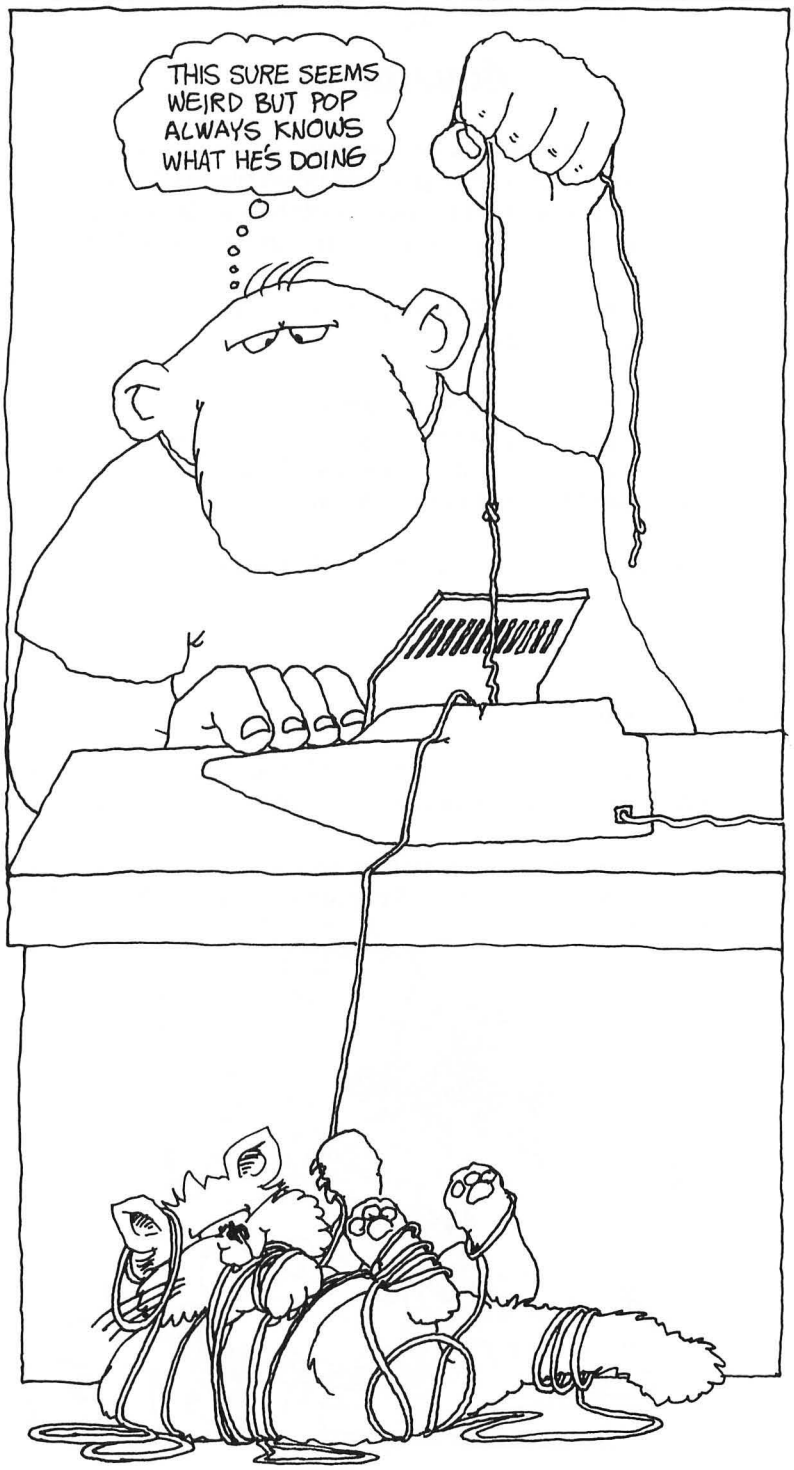
LESS MEMORY: Strings use less memory than any other form of array.

PROTECTION: Strings are automatically placed in a protected area of the computer's memory.

FASTER THAN A SPEEDING BULLET: Strings load data into your program faster than any other method.

STRING IS FUN!: The Wacko Cats like to play with string.

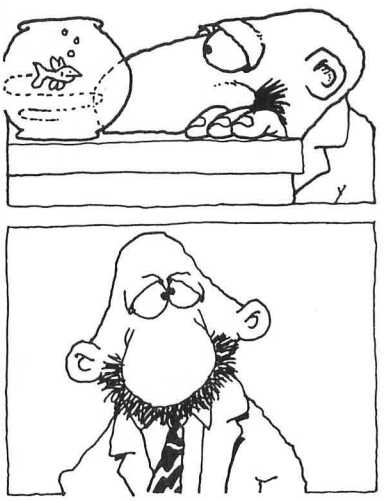



THIS SURE SEEMS WEIRD BUT POP ALWAYS KNOWS WHAT HE'S DOING

5
Movement

When I first began my career as the world's foremost computer aracade game designer, I soon got tired of simple flip-flop animation. I wished that I could get the Jogger off his treadmill and onto the track, so to speak. I wanted to control the Jogger's destiny.

I knew that I had to find a way to make my arcade game characters move. But how?



First I studied movement. I spent weeks observing how things moved. I watched cars speed past my house, the Wacko cats run in circles chasing their tails, and Junior slinking out of the house every time I asked him to take out the garbage.

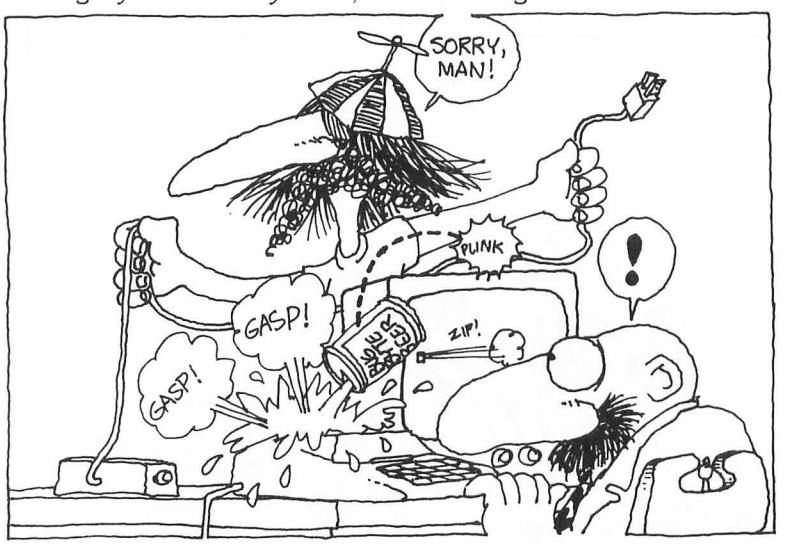
I became cross-eyed. Just look at this old photo:

Sad, isn't it?

A Revelation!

Out of desperation I called in my friend Captain Action, and both of us became cross-eyed staring at my computer's monitor.

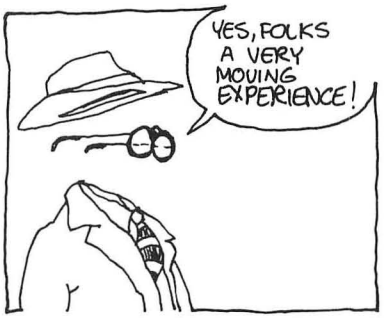
Then one day, when Captain Action accidentally spilled his can of Bug Byte on the keyboard, it all came together. Just before the



computer gasped its final breath, the cursor whizzed diagonally across the screen! *REVELATION!* And a broken computer. But WE DID IT! We moved something across the screen!

That brief flash of movement was etched indelibly in my mind. I spent nights dreaming about the implications of the cursor's movement.

A Realization!



Then one night, I realized what it all meant! I realized that because the screen is a two-dimensional surface, *all references to position can be defined in terms of X and Y coordinates*!

To move an object—a cursor, for instance—all I needed to do was add a change factor to the cursor's starting postion. I quickly flipped open my old high school math book and discovered that a change factor is called *delta*.

Now, armed with this knowledge, my movement theory became crystal-clear. All I had to do to move my cursor would be to *continually update its position on the screen*.

I rushed down to my lab and wrote a few simple position updating formulas using the letter "D" to represent delta.

X = X + DX

Translated into English, this means: The new position X equals the old position X plus a change in the X direction (horizontal movement, across the screen).

Y = Y + DY

This equation means: The new position Y equals the old position Y plus a change in the Y direction (vertical movement, up and down the screen).

Next I wrote a simple movement program using these two formulas, and waited nervously for my computer to be repaired.

Well, that was a long, long time ago. My computer was repaired and my movement theories worked perfectly. Here, I'll show you. Enter this short program and we'll go over it together.

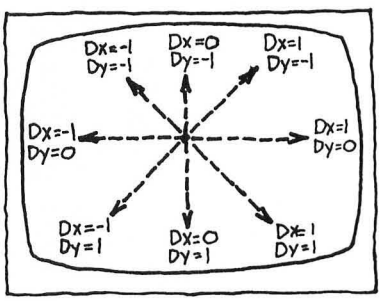
Simple Movement

```
10 GRAPHICS 3
12 .
16 .
20 X = 20:Y = 10:DX = 1:DY = 0
22 .
30 FOR A = 1 TO 10
32 .
40 X = X + DX:Y = Y + DY
42 .
50 COLOR 1:PLOT X,Y
60 FOR PAUSE = 0 TO 250:NEXT PAUSE
70 NEXT A
```

RUN this Simple Movement program and watch what happens. Ready?

An orange line (COLOR 1) moves toward the right from the center of the screen! Cursor movement!

Line 20 first sets the cursor's starting position (X = 20 and Y = 10) close to the center of the screen.



Next it defines the delta or change in each direction: DX = 1 and DY = 0. The cursor will move 1 space at a time in the X direction with no movement at all in the Y direction.

Take a look at my TV screen. Enter new values for DX and DY and make your cursor move in any direction you'd like!

Cursor movement is not limited to the eight directions you see on my TV. Enter the values DX = 2 and DY = −1, and see what happens. These values move the cursor two spaces in the X direction (right) and −1 space in the Y direction (up) every time the program cycles. Get the picture?

Line 40 contains my famous update formulas! Every time the program cycles, the cursor's position is updated by the values we placed in DX and DY in line 20. Line 50 then PLOTs the cursor to the screen.

Line 60 slows down the movement so you can see it. Change 250 to a lower number to make the cursor move faster, or change it to higher number to slow the movement down.

For the benefit of all you aspiring game programmers, I've slightly modified my Simple Movement program by changing a couple of lines and turning it into a utility!

Enter this program, RUN it, and start having some fun. It'll help you see these fundamental concepts in action.

Simple Movement Utility

```
10 GRAPHICS 3
12 PRINT CHR$(125):PRINT "ENTER DX,DY";
16 TRAP 12:INPUT DXM,DYM
20 X = 20:Y = 10:DX = DXM:DY = DYM
30 FOR A = 1 TO 10
40 X = X + DX:Y = Y + DY
50 COLOR 1:PLOT X,Y
60 FOR PAUSE = 0 TO 250:NEXT PAUSE
70 NEXT A
80 GOTO 10
```

It's Easy to Use!

Just enter two values, one for DX and one for DY, separated by a comma. Then press RETURN. Now watch the results!
I wish I had had a program like this when I first studied movement. My eyes wouldn't have crossed and my computer wouldn't have sizzled. (Sigh!)

The Cursor's Behind (Blush!)

Yes, this Simple Movement program demonstrates the fundamentals of movement. But the cursor leaves its image on the screen after it moves.

To make it appear as if the cursor is moving without a behind, you've got to add two more statements to the Simple Movement program:

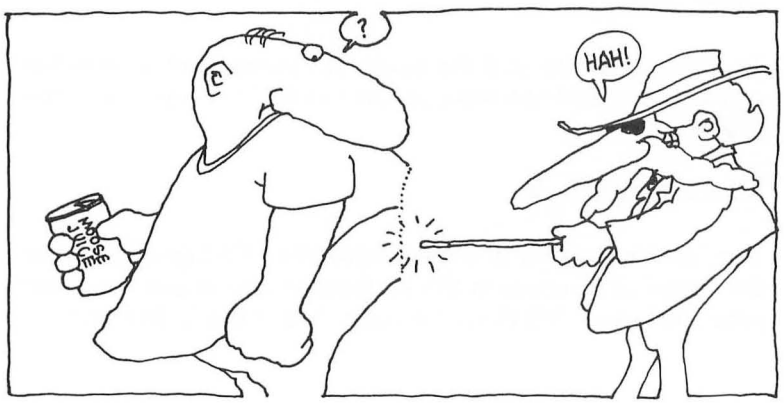
XB = X and YB = Y

Yep, you guessed it. In the next program XB tags along *behind* the cursor as it moves in the X direction and erases the cursor's previous image. YB does the same, but in the Y direction.

This short program shows you what I mean:

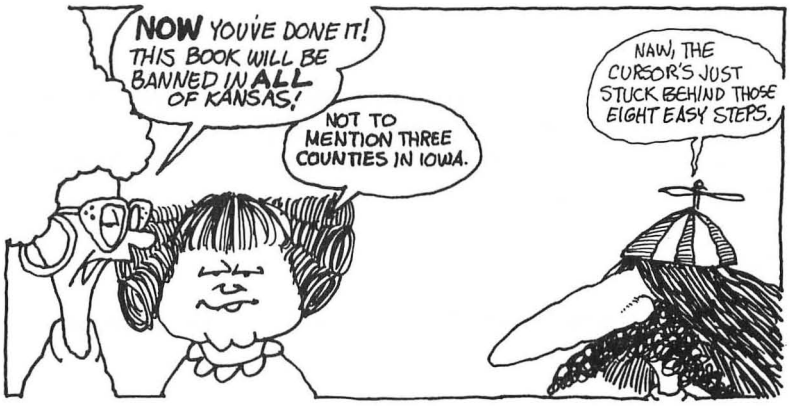
The Erasing Behind

```
 10 GRAPHICS 3
 12 .
 14 . 1. Set up X & Y's starting locations; X & Y's
    movement direction; and set XB & YB to equal X & Y:
 20 X = 20:Y = 10:DX = 1:DY = 0:XB = X:YB = Y
 22 .
 24 . 2. Begin movement cycle:
 30 FOR A = 0 TO 10
 32 .
 34 . 3. Set next X,Y coordinates:
 40 X = X + DX:Y = Y + DY
 42 .
 44 . 4. Erase current X,Y coordinates:
 50 COLOR O
 60 PLOT XB,YB
 62 .
 64 . 5. Plot new X,Y coordinates:
 70 COLOR 1
 80 PLOT X,Y
 82 .
 84 . 6. Set next erase coordinates:
 90 XB = X:YB = Y
 92 .
 94 . 7. Control cursor's movement speed:
100 FOR PAUSE = 0 TO 250:NEXT PAUSE
102 .
104 . 8. Recycle back to line 30:
110 NEXT A
```



Eight Easy Steps to A Firm Behind

First RUN this program. Then read these eight easy steps to get a firm feeling for the cursor's behind.


NOW YOU'VE DONE IT! THIS BOOK WILL BE BANNED IN ALL OF KANSAS!
NOT TO MENTION THREE COUNTIES IN IOWA.
NAW, THE CURSOR'S JUST STUCK BEHIND THOSE EIGHT EASY STEPS.

Here's how this program works, step by step:

Step 1

Line 20: Three "setup" functions take place in this line:

1. X = 20: Y = 10: Sets up the starting coordinates of the cursor.
2. DX = 1: DY = 0: Sets up X and Y's movement directions.
3. XB = X: YB = Y: Sets up XB and YB equal to X and Y.

Now the program is initialized.

Step 2

Line 30: The program's cycle begins with this FOR/NEXT loop.

Step 3

Line 40: X = X + DX:Y = Y + DY This line calculates the next coordinates to which the cursor will be plotted.

Step 4

Lines 50 and 60: The cursor's current X,Y coordinates are erased by plotting the background color (COLOR 0) over it.

Step 5

Lines 70 and 80: The cursor is ploted at new X,Y coordinates. These new coordinates were determined in line 40.

Step 6

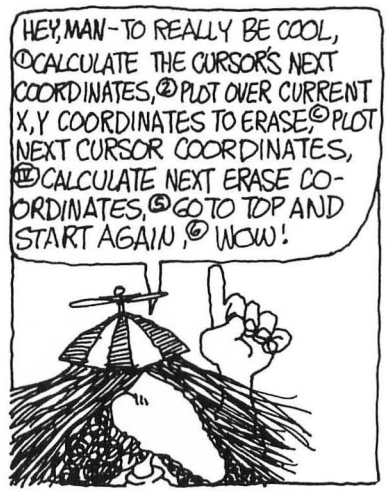
Line 90: XB = X:YB = Y: This line updates the values of XB and YB to equal the cursors currently plotted position. These values will be used in lines 50 and 60 to erase the cursor's tail during the program's next cycle.

Step 7

Line 100: This FOR/NEXT loop is used to control the cursor's movement speed.

Step 8

Line 110: One cycle has been completed and the program goes back to line 30 to begin again.


HEY, MAN - TO REALLY BE COOL, ①CALCULATE THE CURSOR'S NEXT COORDINATES, ② PLOT OVER CURRENT X,Y COORDINATES TO ERASE; ③ PLOT NEXT CURSOR COORDINATES, ④ CALCULATE NEXT ERASE CO-ORDINATES, ⑤ GO TO TOP AND START AGAIN; ⑥ WOW!

Work through this program until you really understand its logic; then modify its elements to see what will happen. Change the cursor's movement speed in line 100; change its direction of movement in line 10; change the cursor's starting location; try other graphics modes. Use your imagination. Make the cursor go where you want it to go!

If you're a real fanatic like Captain Action, add these three weird-looking lines to the program and let the computer show *you* what's going on:

```
65 POKE 752,1:PRINT CHR$(127);CHR$(127);
   "XB = ";XB;" X = ";X:PRINT CHR$(127);CHR$(127);
   "YB = ";YB;" Y = ";Y
105 PRINT CHR$(125)
120 COLOR 0:PLOT X,Y:GOTO 20
```

Making Tracks

The Erasing Behind program works so well that it erases everything the cursor moves over. Sometimes it works too good! There are times when you'd like your character(s) to move over other images on the screen—like buildings or other obstacles—without erasing them!

To see how well the Erasing Behind program tracks across the screen, just add these new lines to the program:

```
 10 GRAPHICS 3:GOSUB 200
110 NEXT A:END
200 FOR X = 20 TO 30
210 COLOR 2
220 PLOT X,10
230 NEXT X
240 RETURN
```

This short modification draws a green line directly in the path of the cursor's movement. When you RUN, this new program the cursor erases the green line.

LOCATE to the Rescue!

Use our old friend LOCATE! It returns the value of the COLOR at the specified X and Y coordinates. We'll use this valuable statement to check the COLOR that's plotted on the screen; then, instead of erasing it, we'll plot this COLOR behind the cursor as it moves.

Here's the complete program:

The Drawing Behind

```
 10 GRAPHICS 3:GOSUB 200
 20 X = 20:Y = 10:DX = 1:DY = 0:XB = X:YB = Y
 22 .
 25 LOCATE X,Y,Z:. FIRST LOCATE STATEMENT
 27 .
 30 FOR A = 0 TO 10
 40 X = X + DX:Y = Y + DY
 42 .
 50 COLOR Z
```

```
 60 PLOT XB,YB
 62 .
 65 LOCATE X,Y,Z:. SECOND LOCATE STATEMENT
 67 .
 70 COLOR 1
 80 PLOT X,Y
 90 XB = X:YB = Y
100 FOR PAUSE = 0 TO 250:NEXT PAUSE
110 NEXT A:GOTO 10
200 FOR X = 20 TO 30
210 COLOR 2
220 PLOT X,10
230 NEXT X
240 RETURN
```

Run this spiffy program! Notice that the cursor does *not* erase the green line.

That's because I've inserted two LOCATE statements into the program. One in line 25, and one in line 65.

The First LOCATE Statement

The first LOCATE statement, in line 25, is used only once. It returns the value of the COLOR at the cursor's starting position. This value (Z) is used in line 50 during the program's first cycle to plot background color at the start position. If you don't include this LOCATE statement, the cursor will write over its starting position.

Remove line 25 and RUN this program to see what I mean.

The Second LOCATE Statement

The second LOCATE statement is used during the balance of the program to plot the background color behind the cursor (at the cursor's previous position).

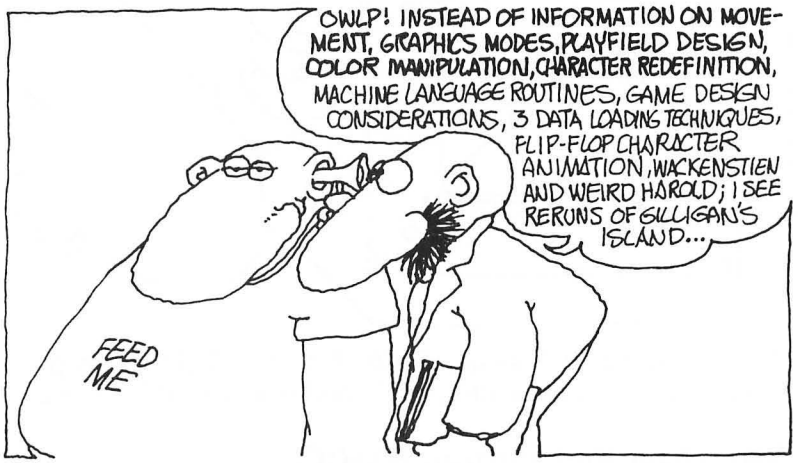
Now that you know everthing about movement, it's time to take control and move the cursor with a joystick!

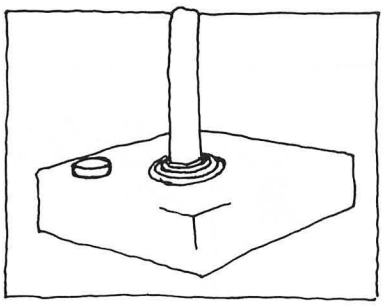
6
Taking Control with Your Joystick

I'm impressed! You'll soon know how to control the cursor, and then characters, with your joystick. After that, it's a short step to developing entire arcade games!

You're at the brink of something big. . . riding the crest of the wave. . . and you're well equipped to continue.

Your bank of knowledge is immense. Your brain is loaded with arcade game design information.


OWLP! INSTEAD OF INFORMATION ON MOVEMENT, GRAPHICS MODES, PLAYFIELD DESIGN, COLOR MANIPULATION, CHARACTER REDEFINITION, MACHINE LANGUAGE ROUTINES, GAME DESIGN CONSIDERATIONS, 3 DATA LOADING TECHNIQUES, FLIP-FLOP CHARACTER ANIMATION, WACKENSTIEN AND WEIRD HAROLD; I SEE RERUNS OF GILLIGAN'S ISLAND...
FEED ME



And this is the miraculous gadget that will bond all these elements together, putting *you* in control.

Joystick Basics

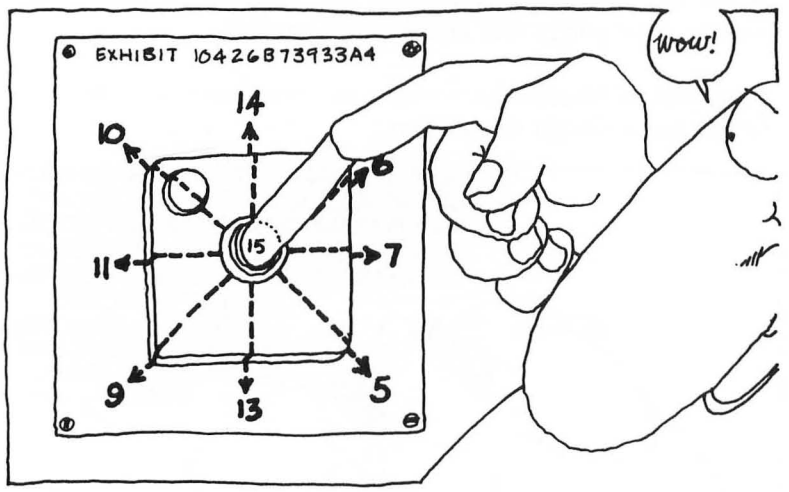
There are either two or four joystick ports located on your Atari computer. Each joystick is refered to in Atari BASIC as a STICK numbered either 0, 1, 2, or 3.

STICK(0) is the joystick that's plugged into port 1.
STICK(1) is the joystick that's plugged into port 2.
STICK(2) is the joystick that's plugged into port 3.
STICK(3) is the joystick that's plugged into port 4.

The joystick generates a specific number depending on the direction it's pushed. You can use this number in your program to control what's happening. Here's how this phenomenon works.

10 PRINT STICK(0): GOTO 10

Enter and RUN this one-line program, plug a joystick into port 1, move it around, and compare the results with this drawing.



They're the same! Change STICK(0) to STICK(1) in this program, plug the joystick into port 2, and watch what happens!

The Little Red Button: STRIG

That little red button, sometimes called a trigger, is refered to in Atari BASIC as STRIG.

STRIG(0) is the trigger of the joystick plugged into port 1.
STRIG(1) is the trigger of the joystick plugged into port 2.
STRIG(2) is the trigger of the joystick plugged into port 3.
STRIG(2) is the trigger of the joystick plugged into port 4.

Enter and RUN this program, plug your joystick into port 1, press and release the button, and watch the results.

110 PRINT STRIG(0):GOTO 10

A value of 0 is returned when the trigger is pressed. The number 1 is returned when the trigger is not pressed.

Play with De Stick and Push De Button

Here's a short program that illustrates joystick control. Enter and RUN it, then play around with your joystick and trigger.

Joystick Control

```
 5 GRAPHICS 18
10 A = STICK(0):B = STRIG(0)
20 POSITION 5,5
30 ? #6;"STICK(0) = ";A;" ";
40 POSITION 5,6
50 ? #6;"STRIG(0) = ";B;" ";
60 GOTO 10
```

Combining STICK and STRIG

As you'll see (when I show you "The Big Frame-Up" in the next chapter), you can combine the STICK and STRIG values to generate up to eighteen numerical outputs. This means that one joystick is capable of controlling up to eighteen program elements, calling up eighteen different characters, for example.

More on this later. For now let's take control of that cursor and start pushing it around!

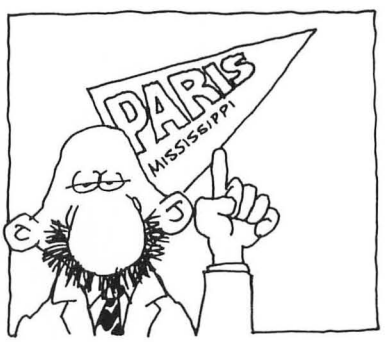
Here's a simple program that puts you in the driver's seat. You're already familiar with most of the concepts shown—I'll explain the new ones—so enter it, RUN it, and enjoy!

Total Control with Bouillabaisse Logic

```
 5 . 1. Select Graphics Mode & set up cursor's start
   locations & change factors
 7 .
10 GRAPHICS 3:X = 10:Y = 10:XB = X:YB = Y
12 .
14 . 2. Assign 'A' to equal STICK(0)
16 .
20 A = STICK(0)
22 .
24 . 3. Bouillabaisse Logic - This tasty concept is
   explained below.
26 .
```

```
30 DX = (A = 6 OR A = 7 OR A = 5) - (A = 11 OR A = 9
   OR A = 10)
40 DY = (A = 9 OR A = 13 OR A = 5) - (A = 10 OR A = 14
   OR A = 6)
42 .
44 . 4. Cursor's next position.
46 .
50 X = X + DX
60 Y = Y + DY
62 .
64 . 5. Make sure that cursor stays within boundries of
   the screen.
66 .
70 IF X>39 OR X<0 THEN X = X - DX
80 IF Y>23 OR Y<0 THEN Y = Y - DY
82 .
84 . 6. Erase cursor's current position.
86 .
90 COLOR 0:PLOT XB,YB
92 .
94 . 7. Plot cursor's new position.
96 .
100 COLOR 1:PLOT X,Y
102 .
104 . 8. Set XB & YB to equal cursor's position.
106 .
110 XB = X:YB = Y
112 .
114 . 9. Return to beginning of program for next cycle.
116 .
120 GOTO 20
```

Bouillabaisse Logic



When I was in Paris . . . that's Paris France, not Texas . . . sojourning at the Sorbonne, I learned an exciting concept. I think it was called either Boolean Logic or Bouillabaisse Logic. Bouillabaisse sounds better to me. Anyway, here's what it's all about.

Bouillabaisse Logic is a fishy way of pointing the cursor in the direction selected by joystick movement.

```
30 DX = (A = 6 OR A = 7 OR A = 5) - (A = 11 OR A = 9 OR
   A = 10)
```

40 DY = (A = 9 OR A = 13 OR A = 5) – (A = 10 OR A = 14 OR A = 6)

These lines return a value of DX and DY that's either –1, 0, or 1 depending on the direction the joystick is pushed. These values are then used in lines 50 and 60 to determine the cursor's next position.

I'll show you a few examples to give you the idea:

The Cursor Moves Up: DX = 0, DY = –1

If you push the joystick up to move the cursor up the screen, A = 14. (Remember, A = STICK(0).)

The number 14 is not present in line 30, so DX = 0 (DX = 0 – 0). No movement in the X direction (across the screen).

The number 14 is present in the second statement of line 40, so DY = –1 (DY = 0 – 1). Movement occurs in the –Y direction (up the screen).

The Cursor Moves to the Right: DX = 1, DY = 0

If you push the joystick to the right to move the cursor toward the right of the screen, A = 7.

The number 7 is present in the first statement of line 30, so, since the values returned are 1 and 0, DX = 1. Movement occurs in the X direction (toward the right of the screen).

The number "7" is not present in line 40, so DY = 0, since DY = 0 – 0. No movement occurs in the Y direction.

The Cursor Moves Diagonally Down and Right: DX = 1, DY = 1

If you push the joystick diagonally down toward the right to move the cursor diagonally down toward the right of the screen, A = 5

The number 5 is present in the first statement of line 30, so DX = 1 (DX = 1 – 0). Movement occurs in the X direction (toward the right of the screen).

The number 5 is present in the first statement of line 40, so DY = 1 (DY = 1 − 0). Movement occurs in the Y direction (towards the bottom of the screen).

Use this piece of scratch paper to take care of a persistent itch, or to work through the other six joystick/cursor movement combinations.

Lines 70 and 80: These two lines are used to ensure that the cursor doesn't go past the boundaries of the screen. These expressions are set for graphics mode 3, which has a screen size of 40X24. If you use a different graphics mode, change these numbers to equal 1 less than the new screen size.

Here's How They Work. . .

IF the cursor bumps into the screen's boundary THEN movement is stopped by making the cursor move one position away from the boundary. The rest of the program is Old Hat

You know all the other elements that give you total control. If you're unclear about anything, review the Movement section. If you still need help, see me after class!

Mess 'Em Up

There's no reason why the cursor has to go to the right when you push the joystick toward the right. You can make the cursor do all sorts of weird things. Just to create mayhem, insert these two lines in place of lines 30 and 40, RUN the program, and stand back!

```
30 DX = (A = 11 OR A = 10 OR A = 6) - (A = 9 OR A = 7
   OR A = 10)
40 DY = (A = 13 OR A = 14 OR A = 5) - (A = 10 OR A = 11
   OR A = 7)
```

De Little Button

It's now time to get the little red button into the act. Just make these two simple changes to the Total Control program.

1. Get the trigger involved by replacing line 20 with:

20 A = STICK(0):B = STRIG(0)

2. Add this line:

35 IF B = 0 THEN DY = DX:DX = 1:GOTO 50

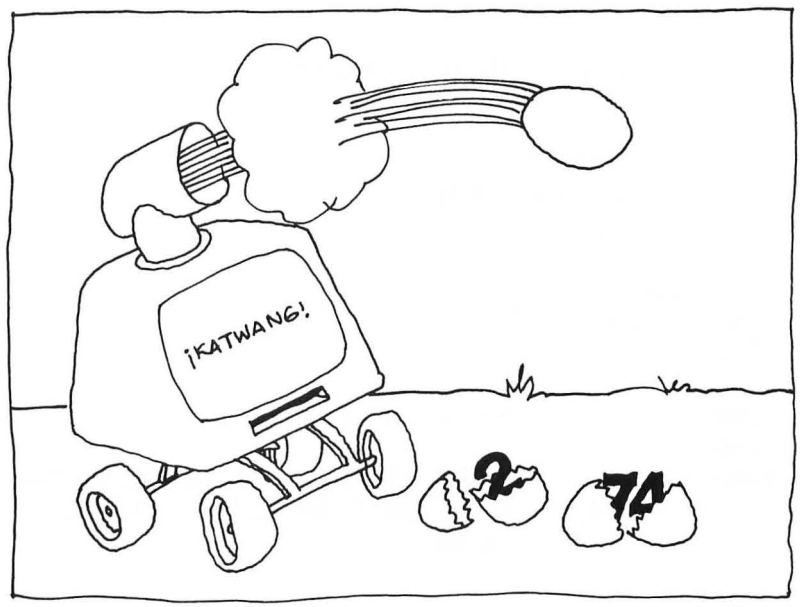
Now, when you RUN your program and press the button, the cursor will whiz across the screen. AMAZING! TOTAL, TOTAL CONTROL.

Faster Control: The Machine-Language Machine's Joystick Movement Routine

The Machine-Language Machine developed this superroutine that makes joystick interpretation easier and movement on the screen up to 30 percent faster than standard BASIC movement:

104, 104, 104, 170, 104, 189, 120, 2, 140, 176, 2, 74, 74, 41, 3, 56, 233, 2, 16, 2, 169, 2, 133, 212, 169, 0, 133, 213, 96

I know, just another bunch of numbers! But these 29 magical numbers will vastly improve the look and speed of your arcade games.



No More Fishy Movement!

The Bouillabaisse Logic routines are replaced with USR routines in the following program. Although the bouillabaisse routines tasted great, they weren't efficient enough for the ever-hungry Dr. C. Wacko! (Burp!)

Superfast Control: Featuring USR Routines!

```
 10 GRAPHICS 3:GOSUB 130:X = 10:Y = 10:XB = X:
    YB = Y
 20 .
 24 .USR Joystick Movement Routines
 26 .
 30 DX = USR(SL,0,0) - 1
 40 DY = USR(SL,0,1) - 1
 42 .
 44 .
 46 .
 50 X = X + DX
 60 Y = Y + DY
 70 IF X<0 OR X>39 THEN X = X - DX
 80 IF Y<0 OR Y>23 THEN Y = Y - DY
 90 COLOR 0:PLOT XB,YB
100 COLOR 1:PLOT X,Y
110 XB = X:YB = Y
120 GOTO 30
122 .
124 .Load Joystick Machine Language routine into Page 6,
    Free RAM
126 .
130 RESTORE 1000
140 FOR A = 1536 TO 1564
150 READ B
160 POKE A,B
170 NEXT A
180 SL = 1536
190 RETURN
192 .
194 .Joystick Machine Language routine.
196 .
1000 DATA 104, 104, 104, 170, 104, 189, 120, 2, 40, 176,
     2, 74, 74, 41, 3, 56, 233, 2, 16, 2, 169, 2, 133, 212,
     169, 0, 133, 213, 96
```

Lines 130 to 180: These lines READ the machine-language DATA into memory locations 1536 to 1564 and make Starting Location (SL) equal to 1536. This is the same stuff we did in the Machine-Language Flip-Flop Jogger program on Page 73 . . . remember?

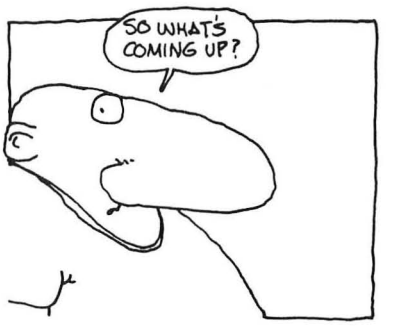
Lines 30 and 40: Here's were all the action takes place! SL is the Starting Location (in memory) of the machine-language routine. The 0s in both lines call out the proper STICK. In this program we're using STICK(0). If your game uses STICK(1), for example, change both 0s to 1s.

The number "0" in line 30 returns the DX value. The number "1" in line 40 returns the DY value. Don't change these numbers!

If left alone, these USR routines would assign one of three values to DX and DY; 0, 1, or 2. These numbers don't work in the program! I've tacked the -1 to the end of each USR statement so it returns -1, 0, or 1, just what the program ordered!

That's all there is to it! Now that you know all about joystick control, you're ready to take the final step: The Big Frame Up, joystick movement control of a multiframe character and Weird Harold and Wackenstein's ignominious return! Eeeeeyech!

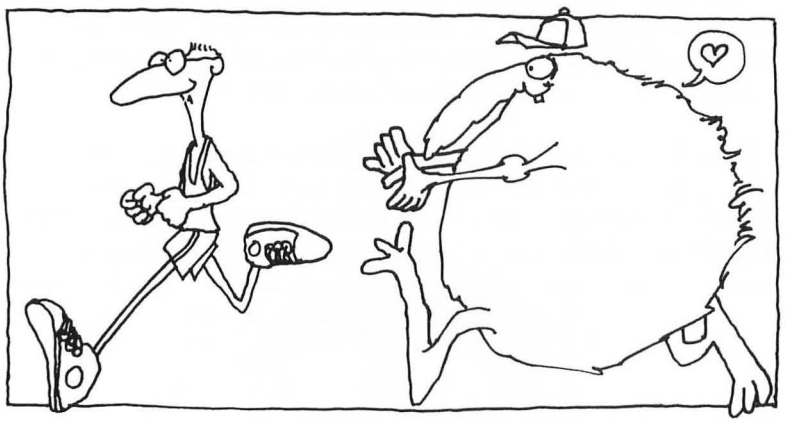
7
The Big Frame-Up: Joystick-Controlled Animated Characters



If you flipped over flip-flop animation and enjoyed pushing that cursor around the screen, you'll go absolutely zonkers over what's coming up next!

After reading and playing with this exciting chapter, you'll be able to push a fully animated character around the screen with your joystick!

The Jogger's finally going to get off his treadmill, and Wackenstein will race wildly across the screen, all under your control.



Let's give that Jogger some exercise first; then we'll plug Wackenstein into our program and turn him loose.

So load up your trusty Monster Maker and we'll get started.

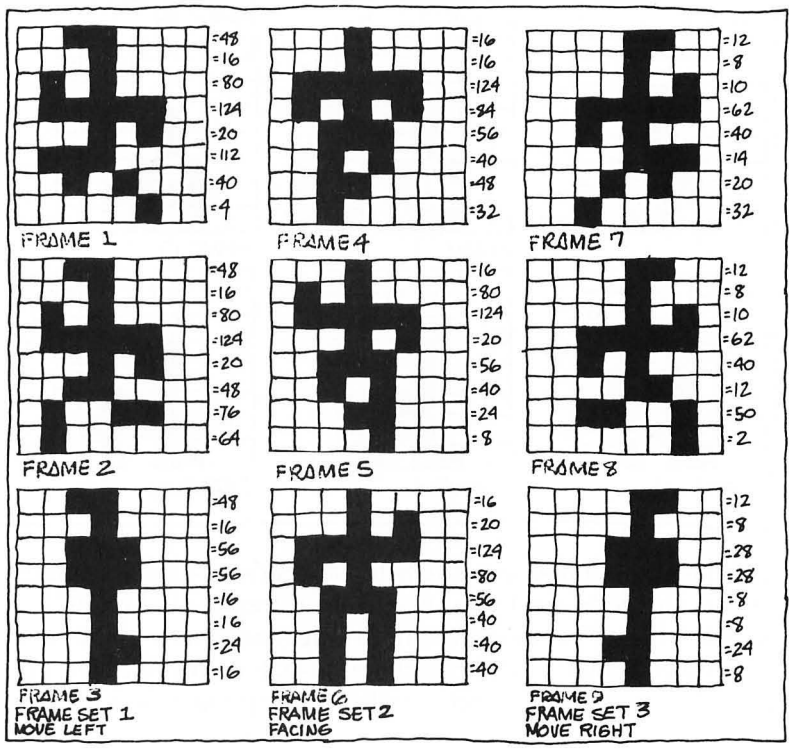
Move That Jogger

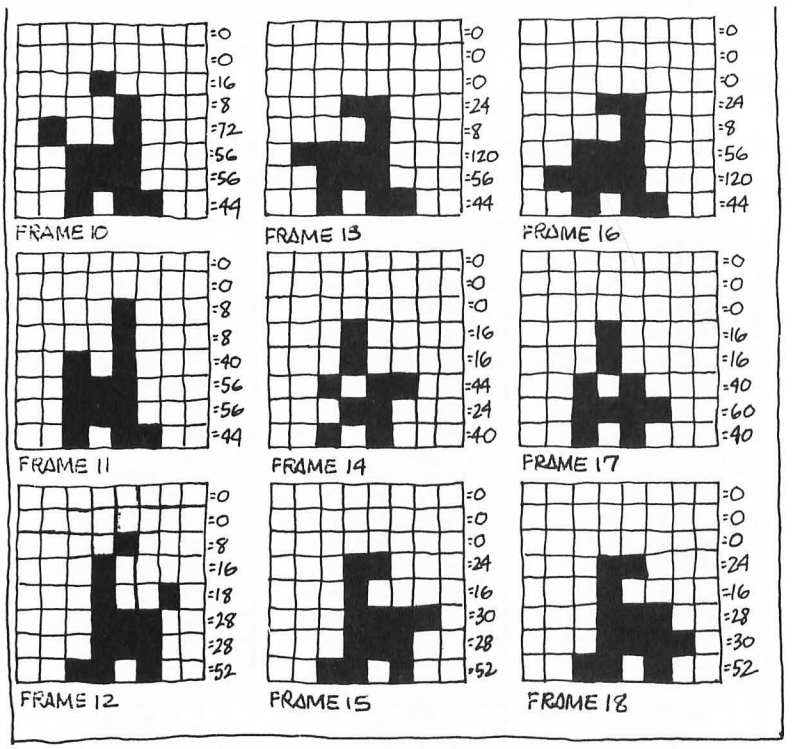
We're going to design a program that'll move an animated jogger about the screen in any direction.

When he runs to the left, he'll face left; when he runs up and down the screen, he'll face you; and when he jogs to the right, he'll face right.

The Jogger is a three-frame animated character. . . so three animation frames are needed for each direction he'll move in, a total of nine frames.

Because he'll be able to move in three directions—left, facing you, and right—we'll design three sets of frames.





Use the Monster Maker to create the Jogger's nine frames plus the extras. Create frame 1 in LOCATION 1, frame 2 in LOCATION 2, and so on. It's very important to enter each frame in the order I've shown above. Entering each frame in its correct sequence (1, 2, 3, 4, 5, 6, 7, 8, 9. . . 18) is critical to the program's operation.

Call It Jogger and SAVE It!

The Big Frame-Up program is designed to GET Jogger animation data from a font file that you've saved to disk or cassette. When you're finished creating the nine Jogger frames plus nine extras, assign them the name JOGGER and SAVE them to disk or cassette.

If you would like to enter the Jogger's frames from data statements, modify lines 210 through 300 of the Big Frame-Up and add eighteen lines of DATA to the program. (Review the Building Block #1 program on page 54)

Enter The Big Frame-Up, plug in your joystick, RUN it, and go absolutely zonkers!!

Important Notice of Great Importance

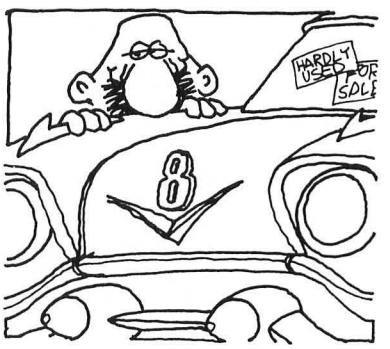
SAVE this program before you RUN it! We'll be making a few modifications to it later!

The Big Frame-Up

```
10 GOSUB 200
12 .
14 . 2. Turn on graphics mode 18 (2 + 16), POKE the
   screen blue, and activate new character set.
16 .
20 GRAPHICS 18:POKE 712,148:POKE 756,START/256
22 .
24 . 3. Set X,Y start locations and set XB & XY equal to
   X and Y.
26 .
30 X = 5:Y = 5:XB = X:YB = Y
32 .
34 . 4. Keep track of frames...The JOGGER uses three
   frames to move in each direction.
36 .
40 FR = FR + 1:IF FR>3 THEN FR = 1
42 .
44 . 5. Let A equal STICK(0).
46 .
50 A = STICK(0)
52 .
54 . 6. Bouillabaisse Logic.
56 .
60 DX = (A = 6 OR A = 7 OR A = 5) - (A = 11 OR A = 10
   OR A = 9)
70 DY = (A = 9 OR A = 13 OR A = 5) - (A = 14 OR A = 10
   OR A = 6)
80 . *Caution. This line is RESERVED*
82 . 7. Character's next position.
84 .
90 X = X + DX:Y = Y + DY
92 .
94 . 8. Make sure that character stays within boundaries
   of screen.
96 .
100 IF X<0 OR X>19 THEN X = X - DX
110 IF Y<0 OR Y>11 THEN Y = Y - DY
112 .
```

```
114 . 9. Erase character's current position.
116 .
120 COLOR 32:PLOT XB,YB:. ATASCII Code 32 is a
    BLANK SPACE!
122 .
124 . 10.****This is the BIGGIE! I'll explain it below!****
126 .
130 COLOR 3+FR+DX*3:PLOT X,Y
132 .
134 . 11. Set XB & YB equal to the character's position
136 .
140 XB=X:YB=Y
142 .
144 . 12. Return for next cycle.
146 .
150 GOTO 40
160 . *These lines (160-190) are RESERVED for some
    big shot*
170 .
180 .
190 .
194 . 1. Make room for character's frames and GET the
    JOGGER font from disk and load it into program.
196 .
200 GRAPHICS 0:START=(PEEK(742)-2)*256
210 CLOSE #1
220 OPEN #1,4,0,"D:JOGGER.FNT"
230 GET #1,CHARACTERS
240 FOR LOCATION=0 TO CHARACTERS-1
250 FOR BYTE=0 TO 7
260 GET #1,SHAPE
262 .
264 . Look at the BIG 8...See it?
266 .
270 POKE START+LOCATION*8+BYTE+8,SHAPE
280 NEXT BYTE
290 NEXT LOCATION 300 CLOSE #1 310 RETURN
```

The Big 8!



I hope you've been paying attention. Look at line 270 again. In the expression BYTE+8, 8 is the offset for the ATASCII character 33—my favorite—the exclamation point!!! Get the point??? When you add 8 to this statement, the Joggger's first frame (as he skedaddles to the left) will be a redefined "!".

If I hadn't added 8 to the expression, the Jogger's first frame would have started at, mercy me, a blank space. No kidding. look it up in the ATASCII chart. Starting out nowhere would have been disastrous. Wheew, defied death once again. You can leave out the 8 if you like to live dangerously.

The Biggie: COLOR 3 + FR + DX * 3:PLOT X,Y

Line 130 is molto, molto, molto importante! (Espresso, anybody?)

Remember when I told you to enter each frame in its correct sequence? Now I'll tell you why.

This line selects and PLOTs the correct frame set (1, 2, or 3) depending on the direction you move your joystick.

Here's how it works:

If you bend the joystick to the left, my Bouillabaisse Logic statement on line 60 makes DX = −1. Work through this simple formula and you'll discover that COLOR 1 will be PLOTed at X,Y.

COLOR 1 is the first frame you entered when you designed the Jogger.

All right already, I'll work through this "simple formula."

I'll replace the variables in this formula with numbers so you can see the results.

Move Left

COLOR 3 + 1 − 1 * 3 = 1

Remember, multiplication is performed first!

When the Jogger moves left, COLOR 1 is PLOTed at X,Y. Then the frame counter in line 40 increases by 1 and COLOR 2 is PLOTed at the next X,Y coordinates. The frame counter increases once more and COLOR 3 is PLOTed at the next X,Y coordinates. Finally the frame counter resets to 1 and the process begins again. So we are cycling through our left-facing frames whenever you move the joystick left!

Facing You

COLOR 3 + 1 + 0 * 3 = 4

COLOR 4 begins the next set of three frames (4,5 and 6). Again, moving the joystick up or down cycles through the appropriate frames.

Move Right

COLOR 3 + 1 + 1 * 3 = 7

COLOR 7 begins the next set of three frames (7, 8 and 9).

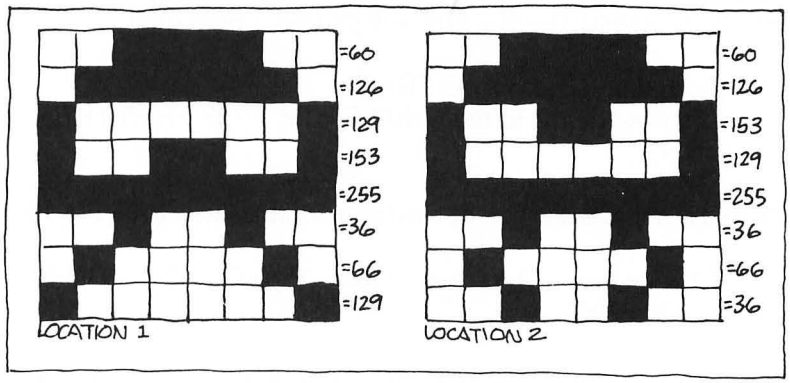
If you entered the frames incorrectly when you first used the Monster Maker to design the Jogger, he might have faced left when he was running to the right, or stopped at the local pub for a brew before finishing the marathon. How totally embarrassing! REALLY!

Phase II

Now that you've got the hang of superpro animation, let's glide into Phase II: animating characters that don't look in the direction they move.

Yes, I'm afraid we're going to experience Phase II, total pixel annihilation, in a mere ten seconds!

STARTING!
10·9·8·7...
BUT, DAD-HOW COME WE GOTTA WEAR THESE WEIRD PROTECTIVE GLASSES?
'CUZ WACKENSTEIN'S COMING TO LIFE AGAIN!
WELL, HE BETTER NOT STAY FOR DINNER!



Make sure that you've designed and saved MONSTER.FNT using the Monster Maker. If you're not sure, just flip back to page 65. When you're ready, make these teeny-weeny changes to the Big Frame-Up program, then RUN it!.

Some Teeny Weeny Changes

1. Change line 40 to read: 40 FR = FR + 1:IF FR>2 THEN FR = 1
2. Change line 130 to read: 130 COLOR FR:PLOT X,Y
3. Line 220: Replace "D:JOGGER.FNT" with "D:MONSTER.FNT"

Wackenstein's a two-frame character (no insult intended). Ergo the change from 3 to 2 in line 40. He's not sophisticated; he dosn't "look" where he's going. As a matter of fact, he bumps into stuff a lot! That's why line 130 is so unsophisticated!

Now It's Weird Harold's Turn!

Weird Harold's apt to get pretty upset if he isn't given the chance to act goofy on the screen. And when Weird Harold gets upset, PEEEUUUU!

We'd better humor him. Turn back to page 81, refer to Weird Harold's six frames and create and SAVE a HAROLD.FNT using the Monster Maker. If you've modified The Big Frame-Up to load Harold's shape from DATA statements, use the data listed in lines 1000 to 1050 of the Amazing Feet program on Page 81.

Ready to modify The Big Frame-Up? O.K., here goes. . . .

1. Add a new line 35: 35 FR = 1.
2. Change line 40 to read: 40 FR = FR + 2:IF FR>5 THEN FR = 1.

Harold is animated in sets of two frames...thats why FR = FR + 2.

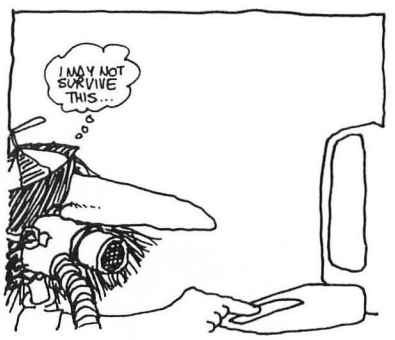
3. Change line 110 to read: 110 IF Y<0 OR Y>10 THEN Y = Y – DY.

Because Harold has amazing feet, he'd go off the screen if we didn't change Y>11 to Y>10.

4 .Add lines 125 and 135 (below) and lines 120 to 135 will look like this:

120 COLOR 32:PLOT XY,YB
125 COLOR 32:PLOT XB,YB + 1
130 COLOR FR:PLOT X,Y
135 COLOR FR + 1:PLOT X,Y + 1

First lines 120 and 125 erase Harold's head and feet. Then lines 130 and 135 draw them. (This sequence is critical. Try it another way, and BARROOOOOM!)



5. Last, but not least, change line 220 to read: OPEN #1,4,0,"D:HAROLD.FNT".

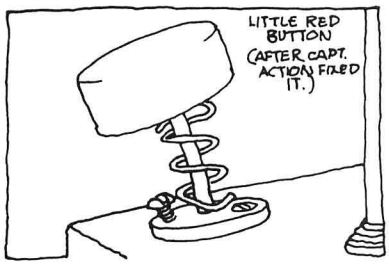
Now stand back, hold your nose, close your eyes, and RUN the program!

That was something, no?

I'm turning into a real computer wacko. I modified lines 35 and 45 and was truly mystified by the results! If you'd like to be mystified,do this:

1. Change line 35 to: FR = 0.
2. Change line 45 to: FR = FR + 2:IF FR = 6 THEN FR = 1.

Now, RUN the program!



A Cautious Cautionary Caution: Beware!

Only make these changes if you are totally unafraid of the far-reaching consequences that wackoness will have on your life-style.

Now I'd like to show you some prestidigitation (magic). Nothing up my sleeves. . . nothing hidden in my pockets—especially money—nothing up here. . . .

Animate Eighteen Frames?

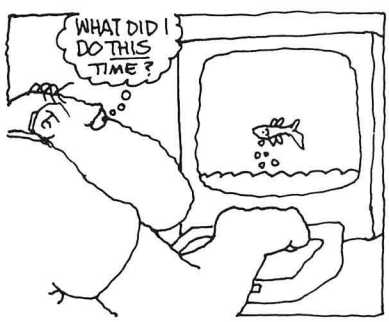
Yes, it is possible folks, with the help of this little RED BUTTON (and a little programming).

If you were paying attention before, you'll have noticed that the JOGGER font contains eighteen animation frames. If you weren't, it doesn't (logical. . . isn't it?).

Watch closely and I'll show you how to make all eighteen frames sway to the joystick's gentle pressure.

First, LOAD the unadulterated Big Frame-Up program, the one that you ran before Wackenstein went berserk and Weird Harold demonstrated his weirdness all over it. Then change line 50 to read: 50 A = STICK(0):B = STRIG(0).

Now, fill up the blank lines I reserved in the program with this sleight-of-hand:




```
 80 IF B = 0 THEN GOTO 160
160 COLOR 32:PLOT XB,YB
170 COLOR 14 + DY * 3 + DX:PLOT X,Y
180 XB = X:YB = Y
190 GOTO 40
```

O.K., now you've done it!

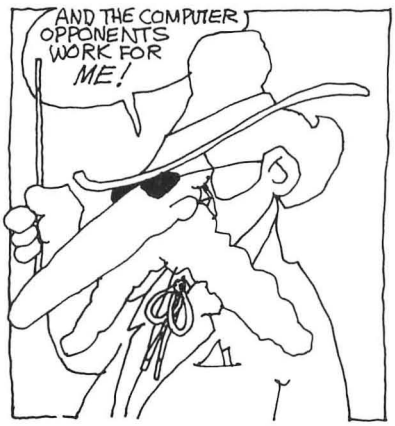
RUN the program and press the trigger while moving the joystick to see exactly what you've done. Then work through lines 160 to 180 to understand why!

A Helpful Hint

Pssst, the same illogical logic I used to explain the Jogger's marathon antics applies to these lines.

One last bit of nonsense. You don't have to begin the first animation frame at OFFSET 1 (ATASCII Code 33). You can begin your Big Frame-Up at any letter's location. If, for example, you want to start with the letter A, just change these three lines as I've done so brilliantly below:

```
130 COLOR 65 + FR + DX * 3 + 2
170 COLOR 76 + DY * 3 + DX + 2
270 POKE START + LOCATION * 8 + BYTE + 264,SHAPE
```



8
Adversaries and Things That Bounce in the Night

Watch out! They're going to get you! Do you get an anxious feeling of urgency when playing games like Centipede™, Missile Command™, Star Raiders™, Pac-Mam, or Atari Bastetball™? You should, because they've all got one thing in common: computer generated opponents!

Computer opponents are heartless creatures whose mission is to challenge your skills and raise your blood pressure while trying their darndest to create confusion and wreak havoc!

A Compendium of Chasers and Scaredy Cats

Many computer opponents are schizophrenic. Their personalities change as you are playing the game. In Pac-Man the ghosts are real meanies, chasing the poor gobbler around the screen until he gobbles up a power pill, then they turn into scaredy cats.

Atari's Basketball's version of Wilt the Stilt chases you when you've got the ball, then turns into a scaredy cat, trying his best to avoid you, when he's got possession. FOUL!

A Colony of Low I.Q.'s

Other computer opponents aren't too bright. They act a little like a school of uneducated fish, avoiding obstacles while swimming mindlessly toward the bait. The centipede, flea, and scorpion in Centipede all act fishy. The centipede, for example, avoids the mushrooms as it winds its way blindly toward the bottom of the screen. When it arrives at the bottom, it starts back up again. All the player has to do is get out of its way, or destroy it.

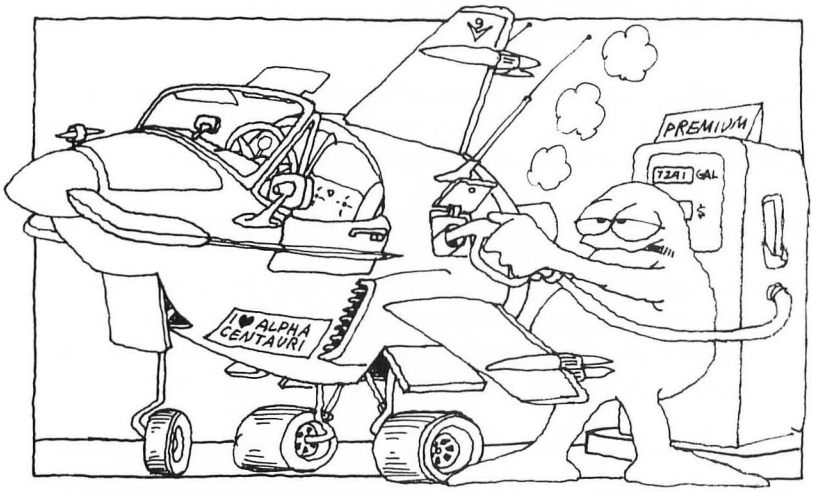
The Chaser-Bouncer Combo

Centipede's spider is another kettle of fish altogether. This dangerous insect chases the player about the screen with less than honorable intentions. If you don't get him, he'll get you! This insect has another sinister trick up one of his eight sleeves; he's a bouncer. If he misses his dinner (you), this creepy character

bounces up and away from the edge of the screen to taunt and confuse you. Frightening, isn't it?

Roving Robots

Star Raiders features what I call roving robots. When your ship is stationary, the enemy vessels circle in front of you in a preprogrammed pattern. They move forward and then change from roving robots to scaredy cats, always keeping their distance.



And Finally, Vandals

These marauders are potentially the most dangerous foes you're apt to face in an arcade game. They're not after you; they're after your possessions! It's your job to stop them before they reach their goal.

In Missile Command, all the missles are "vandals" bent on destroying your cities! Your mission is simple: STOP THEM!

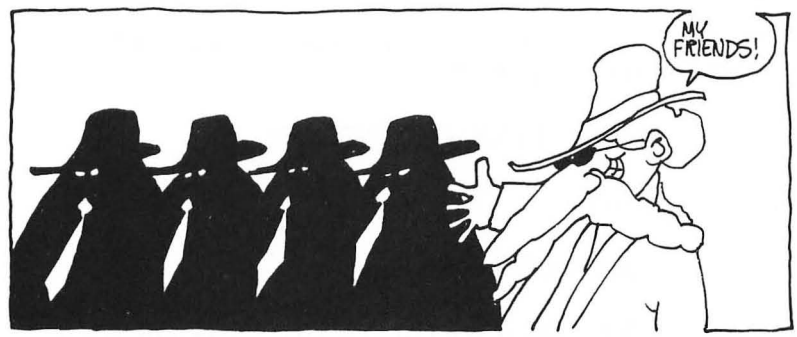
Ever look at the Star Raider's Galactic Map? All those Zylons are heading for your Star Base! Your job is to destroy them before they destroy your bases.

If You Can't Take the Heat, Get Out of the Arcade Game Business

Whew! Not a nice bunch of guys. But , they certainly make an arcade game worth playing, if you can take the heat.

You've mastered some very sophisticated programming techniques during your visit with me, more than enough to design some really action-packed arcade games. And now you know how much excitement computer-generated opponents can add to your games. Interested in learning some of the techniques used by the pros to make them so nasty?

You are? Great! Just read on and venture forth into the cruel and heartless world of C.G.O. . . .



Four Cruel and Heartless Programs

Computer-generated opponents may be heartless, but I'm not. I'm a real cream puff. The following easy-to-use C.G.O. programs are guaranteed to have your players sitting on the edge of their seats.

Most of these programs combine bits and pieces of knowlege you've already studiously acquired. I'm going to have to introduce only a few new programming tricks. From now on it's merely a matter of technique and common (programming) sense.

The Chaser's Out To Get You!

In the Chaser program you'll learn how to develop my world-unknown "Chasem" concept into an engrossing (large?) arcade game.

Here's the listing. I've reserved lines 20, 130, 140, 210, and 220 for future excitment. You reserve them by typing in the line numbers followed by either a period or a REM statement, and we'll fill them in a little later.

The Chaser

```
 10 GRAPHICS 0:POKE 752,1
 20 .
 22 .
 24 . Set up the Chaser's starting locations; and set CXB
    & CYB equal to CX & CY
 26 .
 30 CX = 0:CY = 0:CXB = CX:CYB = CY
 32 .
 34 . Set up the Player's starting locations; and set PXB &
    PYB equal to PX & PY
 36 .
 40 PX = 20:PY = 20:PXB = PX:PYB = PY
 42 .
 50 A = STICK(0)
 52 .
 54 . Good old Bouillabaisse Logic
 56 .
 60 DXP = (A = 6 OR A = 7 OR A = 5) - (A = 11 OR A = 9
    OR A = 10)
 70 DYP = (A = 9 OR A = 13 OR A = 5) - (A = 10 OR A = 14
    OR A = 6)
 72 .
 74 . The Player's next position
 76 .
 80 PX = PX + DXP:PY = PY + DYP
 82 .
 84 . Lines 90 to 120 make the Player appear to leave one
    side of the screen and reappear at the opposite side.
 86 .
 90 IF PX<1 THEN PX = 38
100 IF PX>38 THEN PX = 1
110 IF PY<1 THEN PY = 21
120 IF PY>21 THEN PY = 1
130 .
140 .
142 .
144 . Erase Player's current position.
146 .
150 COLOR 32:PLOT PXB,PYB
152 .
154 . Plot Player's new position. . . the player is ATASCII
    Code 9. . . a plus sign ( + )!
156 .
```

```
160 COLOR 19:PLOT PX,PY
162 .
164 . Set PXB & PYB equal to the Player's current
    position.
166 .
170 PXB = PX:PYB = PY
172 .
174 . Lines 180 to 240. . . La Petite Bouillabaisse and the
    Chaser! I'll tell you how these work below.
176 .
180 DXC = SGN(PX - CX):DYC = SGN(PY - CY)
190 CX = CX + DXC:CY = CY + DYC
200 COLOR 32:PLOT CXB,CYB
210 .
220 .
230 COLOR 42:PLOT CX,CY
240 CXB = CX:CYB = CY
242 .
244 . Return to beginning of program for next cycle.
250 GOTO 50
```

The Chaser program has two characters, you (the Player) and the computer-generated Chaser.

Who's Who?

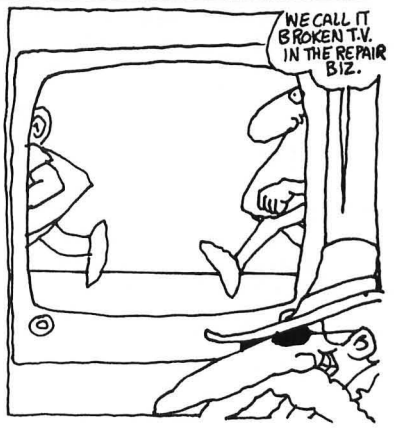
To keep track of who's who in this example, I've ingeniously assigned the letter P to all of the Player's movement statements (like PX) and the letter C to the statements that move the Chaser (like CX).

Familiarity Breeds Familiarity

With the exception of line 40, which sets up the Chaser's starting locations, lines 10 through 170 let you move the Player about the screen with your joystick. Sound familiar?

If you don't believe me, change line 172 to: GOTO 50. RUN the program to see what I mean.

Wrapping Around Lines 90 to 120



Look at lines 90 to 120. Instead of lines 90 to 120 limiting the Player's movement within the screen's boundaries, they make the Player appear to leave one side of the screen, only to reappear at the opposite side. This illusion is called *wraparound* in the arcade biz.

Line 180: La Petite Bouillabaisse

In lines 60 and 70 the Player's X,Y joystick movement is converted to either -1, 0, or 1. SGN in line 80 does the same for the computer-generated Chaser!

Give Me a Little SGN

SGN is a little-known BASIC instruction that, thank goodness, has nothing whatsoever to do with trigonometry. It is not sine/cosine stuff!

When you use SGN, as I have in line 180, it makes DXC equal -1, 0, or 1 depending on the results of the simple math within the parentheses.

SGN In, Mystery Guest!

Here are a few examples that show how SGN works. Enter these three examples in the "immediate mode" (no line numbers) and hit RETURN to see the amazing results:

1. **DXC = SGN(1 – 1) <RETURN>**

The answer is 0! the numbers are equal and the subtraction results in 0.

2. **DXC = SGN(5 – 3) <RETURN>**

The answer is positive 1 because $5-3=2$ and 2 is a positive number.

3. **DXC = SGN(3 – 5) <RETURN>**

The answer is -1 because $3-5=-2$ and -2 is a negative number.

Try other combinations and mathematical expressions. You'll discover that if the answer is negative, SGN returns −1. If the answer is positive, SGN returns 1. And, if the answer is zero, SGN returns 0.

Total Understanding of the Chasem Concept!

The change in the Chaser's X and Y positions (DXC and DYC in line 180) is determined by the Player's position minus the Chaser's positon: PX−CX and PY−CY. These values of change (−1, 0, or 1) are used in lines 190 through 240 to make the Chaser persistently do what it's best at—chase you!

A Full-Fledged Game, Almost

Now that you understand this concept completely, let's fill in those reserved lines with a couple of LOCATE statements and other stuff, and develop a full-fledged arcade game.

Olly, Olly, In Come Free!

The goal of this simple game is to reach your Star Base before the enemy Chaser catches you. It's modeled after the game Tag.

Here's How to Program Your Game

1. First, add line 20 to plot the Star Base on the screen:

20 COLOR 32:PLOT 2,0:COLOR 16:PLOT 3,1

Color 32 (blank space) at locations 2,0 erases that white cursor in the upper left corner of the screen. Color 16 (a "clubs" symbol) plots the Star Base at locations 3,1.

2. Now, add lines 130 and 140:

```
130 LOCATE PX,PY,Z
140 IF Z = 16 THEN POKE 710,195:POSITION
    13,10:PRINT "YOU'RE A WINNER!":FOR A = 1 TO
    1000:NEXT A:RUN
```

When the Player touches the Star Base the LOCATE statement returns a value of 16 (its COLOR), the screen turns green since Z = 16, and YOU'RE A WINNER. Then, after a short pause, the game begins again.

3. Finally, add lines 210 and 220:

```
210 LOCATE CX,CY,Z
220 IF Z = 19 THEN POKE 710,53:POSITION
    15,10:PRINT "GOTCHA!": FOR A = 0 TO 500:
    NEXT A:RUN
```

If the Chaser catches you, LOCATE returns 19 (that's your COLOR—see line 160) the screen turns red and "GOTCHA!" appears. Then, after a brief pause, the game begins again.

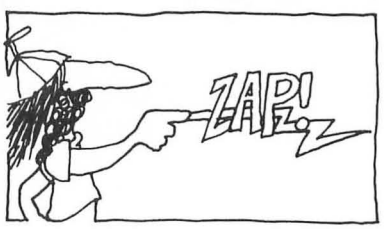
Make Chaser into a Real Blockbuster, Buster

Build on this simple game. Use your imagination and arcade game programming skills to make Chaser into a real blockbuster smash hiteroo!

Six Brilliant Ideas!

1. Redesign Chaser in graphics mode 1 or 2 and use character graphics to create an animated Chaser and Player. To bring this game to life you might want to model your Player on the Jogger. Your Chaser might be Wackenstein or, heaven forbid, Weird Harold!

2. Add sound to the game (see chapter 9). when you are caught—BAROOM! When you win—TAH! TAH!



3. Give your player the ability to shoot back at the Chaser. ZAP!

4. Add scoring to the screen—points for the computer and points for you!

5. Make this a two-player game. Chase each other around the screen.

6. Go outside and play soccer or fly a kite instead!

This Is Your BIG Chance!

This may be your BIG chance to design a revolutionary new arcade game. Don't pass up this great opportunity! Have fun, show off, and put all your knowledge to work! (Before it's too late.)

The Scaredy Cat, Starring the Three Wacko Cats

The Scaredy Cat program is very similar to the Chaser. I'll point out the major differences after you type it in and RUN it.

Scaredy Cat

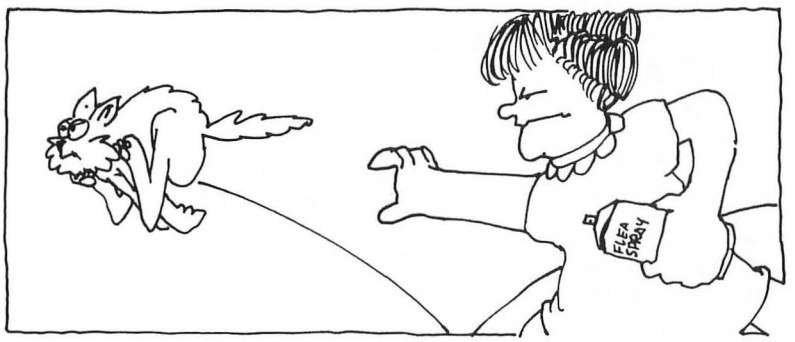
```
 10 GRAPHICS 0:POKE 752,1
 20 COLOR 32:PLOT 2,0
 30 CX = 0:CY = 0:CXB = CX:CYB = CY
 40 PX = 20:PY = 20:PXB = PX:PYB = PY
 50 A = STICK(0)
 60 DXP = (A = 6 OR A = 7 OR A = 5) - (A = 11 OR A = 9
    OR A = 10)
 70 DYP = (A = 9 OR A = 13 OR A = 5) - (A = 10 OR A = 14
    OR A = 6)
 80 PX = PX + DXP:PY = PY + DYP
 85 .
 90 IF PX<1 OR PX>38 THEN PX = PX - DXP
100 IF PY<1 OR PY>22 THEN PY = PY - DYP
105 .
110 COLOR 32:PLOT PXB,PYB
120 COLOR 19:PLOT PX,PY
130 PXB = PX:PYB = PY
135 .
140 DXC = SGN(PX - CX):DYC = SGN(PY - CY)
150 CX = CX - DXC:CY = CY - DYC
155 .
160 REM IF CY = PY THEN CY = PXB
170 REM IF CX = CX THEN CX = PYB
175 .
180 IF CX<1 OR CX>38 THEN CX = CX + 4*DXC
190 IF CY<1 OR CY>22 THEN CY = CY + 4*DYC
195 .
200 COLOR 32:PLOT CXB,CYB
210 COLOR 42:PLOT CX,CY
220 CXB = CX:CYB = CY
230 GOTO 50
```

Why the Scaredy Cat's Scared

Lines 90 and 100: These two lines set the boundaries for the Player. In the Scaredy Cat program, the Player is limited by the screen's borders; no illusions this time. Lines 140 and 150:

Because this Cat runs away from the Player instead of chasing after him, minus signs in line 150 replace the plus signs that were used in line 190 of the Chaser program.

Lines 180 and 190: These two lines set the boundaries for the Scaredy Cat. If I didn't put them in, he'd get so scared he'd run off the edge of the screen. Every time the Scaredy Cat bumps against the edge of the screen, the statements CX = CX + 4 * DXC and CY = CY + 4 * DYC make him bounce off by four positions. If I didn't add 4 * to this statement, the Scaredy Cat would stick in the corner! Increase this number if you'd like to see the Cat bounce further away from the wall, or remove 4 * altogether and see what happens.



Lines 160 and 170: The REM statements in front of these two lines are just waiting to be removed. I put them there to show you that the Scaredy Cat *can* be caught. Remove REM, RUN the program, and it becomes impossible to catch him!

Every time the Cat's position equals the Player's, the Cat's position is changed to the Player's previous position. Clever, isn't it?

Schizoid!

Now that you know how the Chaser and Scaredy Cat work, you can put either one, or both, into your arcade games.

Here's an Idea! Modify the Chaser game so that the Chaser's a real schizoid. Make this cruel and heartless enemy turn into a real pussy cat. For example, you might add another Star Base. IF the Player touches it, the Chaser goes flippo, turns into a Scaredy Cat for a specific number of cycles, then reverts back to its cruel and heartless self. You're on your way to designing the next best-seller!

Things That Go BOUNCE in the Night

Before a screen object can bounce off something, your program's got to know that it bumped into something. Stuff on the screen isn't "smart," and the normal laws of physics ("for every action there is an equal and opposite action") don't apply to screen images, you've got to supply the brains and physical laws. Don't panic! The great Dr. Wacko will reveal all!

Colliding Collisions

You know two ways to "tell" the rest of your program that an object has hit something:

1. The LOCATE statement: When the object (ball, spaceship, umbrella) bumps into a screen barrier (wall, closed door, potato chip) the COLOR number returned in the LOCATE statement activates a routine in your program that makes the object reverse its direction.

Remember the Bong program? It's on page 42. Look it over now to refresh your memory. In Bong, when the ball slams into one of the walls a LOCATE statement "tells" it to bounce away.

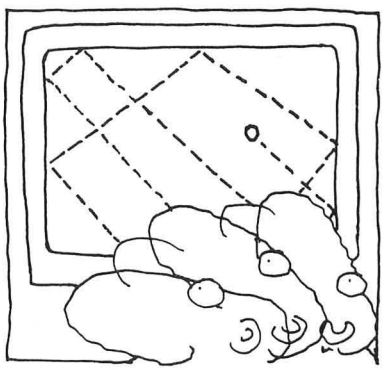
2. Predefined Boundaries: When the object reaches predefined coordinates, an IF/THEN statement is used to reverse its direction. Lines 90 and 100 of the Scaredy Cat program brilliantly illustrate the crystaline simplicity of this technique.

Here's a simple program that uses a LOCATE statement to "tell" the ball when to bounce away from a wall. Enter it, RUN it, then I'll review it with you below.

Locate Bounceroo

```
10 GRAPHICS 5+16:GOSUB 100
20 X=30:Y=20:XB=X:YB=Y:DX=1:DY=1
30 X=X+DX:Y=Y+DY
32 .
40 LOCATE X+DX,Y,Z:IF Z<>0 THEN DX=-DX
50 LOCATE X,Y+DY,Z:IF Z<>0 THEN DY=-DY
52 .
60 COLOR 0:PLOT XB,YB
70 COLOR 1:PLOT X,Y
```

```
 80 XB = X:YB = Y
 90 GOTO 30
 92 .
100 COLOR 2
110 PLOT 25,10
120 DRAWTO 45,10
130 DRAWTO 45,30
140 DRAWTO 25,30
150 DRAWTO 25,10
160 RETURN
```



What Makes Bounceroo Bounce?

First this program GOSUBs in line 100 after selecting the graphics mode, draws a box, and RETURNs.

Next the starting X,Y coordinates of the "ball" are set within the box.

Then, the two LOCATE statements appear.

1. The LOCATE statement in line 40 makes the "ball" bounce away from either of the two vertical walls of the box.
2. The LOCATE statement in line 50 makes the ball bounce away from either of the two horizontal walls of the box.

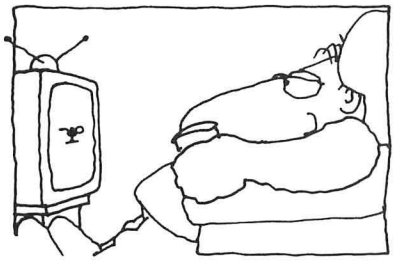
The rest of this program is similar to the many simple movement programs I showed you in chapter 5. Review it if you have any questions, then move on to the second Bounceroo technique, predefined boundaries.

Boundary Bounceroo

Ladies and gentlemint, by changing just two minuscule lines—yes, that's all, just two lines—you, too, can transform the Locate Bounceroo into a Boundary Bounceroo! Amazing? No, just easy, and here's how to do it.

Replace lines 40 and 50 with these two new lines:

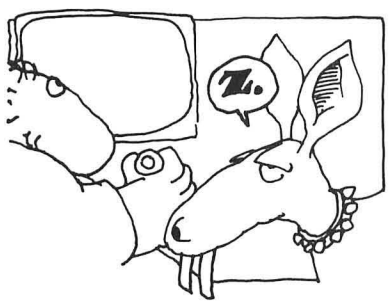
```
40 IF X + DX = 45 OR X + DX = 25 THEN DX = - DX
50 IF Y + DY = 10 OR Y + DY = 30 THEN DY = - DY
```



UM0s: Unidentified Movable Objects

Now RUN your modified program. The ball bounces away from the boundaries of the box! The effect is the same as achieved in the Locate Bounceroo program!

The Boundary Bounceroo program contains the same method I use to keep players and other Unidentified Movable Objects restricted within the boundaries of my playing field.



"I'm not sssccaaarrrredd!"

Now that you know how to design a Chaser and a Bouncer, try combining the two to create your own Super-Meany—like the Centipede's bouncing spider. I'll be back after you've completed this experiment. Not that I'm scared, but. . . .

Roving Robots

One way to design a Roving Robot is to "read" its programmed positions from DATA statements. Here, I'll show you. . .

Roboteroonie

```
10 GRAPHICS 3:GOSUB 120
20 RX = 5:RY = 5:RXB = RX:RYB = RY:RC = 1
22 .
24 . 2) DXR & DYR will both equal 0 when RX & RY
     Robot Coordinates equal the two data statements.
26 .
30 DXR = SGN(RX - A(RC))
40 DYR = SGN(RY - A(RC + 1))
42 .
44 . 3) When both DXR & DYR equal 0 the next set of
     data is selected.
46 .
50 IF DXR = 0 AND DYR = 0 THEN RC = RC + 2
52 .
54 . 4) When last Robot coordinate is reached, begin
     again at first Robot coordinate.
56 .
60 IF RC>8 THEN RC = 1
62 .
70 RX = RX - DXR:RY = RY - DYR
80 COLOR 0:PLOT RXB,RYB
```

```
 90 COLOR 1:PLOT RX,RY
100 RXB = RX:RYB = RY
110 GOTO 30
112 .
114 . 1) Dimension Array, then set up Array read from
    data.
116 .
120 DIM A(8)
130 RESTORE 190
140 FOR A = 1 TO 8
150 READ B
160 A(A) = B
170 NEXT A
180 RETURN
190 DATA 10,10
200 DATA 20,5
210 DATA 0,12
220 DATA 5,5
```

The Roboteroonie program uses many elements you've already learned but in new and exciting ways.

First, the DATA in lines 190 to 220 is put into an array, A(RC), in line 160.

RC is my way of saying "Robot's Coordinate." In Line 20, RC = 1, so the Robot first appears at the coordinates called out by the first set of data (10,10).

Lines 40 and 50: Here's another application of La Petite Bouillabaisse! DXR and DYR will both equal 0 when the Robot's coordinates equal the DATA. Substitute numbers for the symbols in these two formulas and do the simple math to see what I mean.

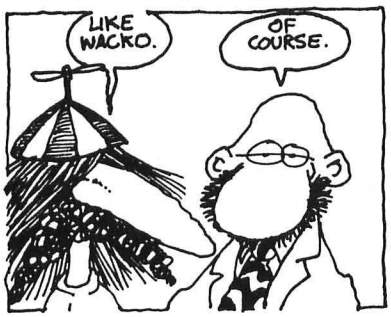
In line 50, when both DXR and DYR equal 0, the next set of data is selected, and the Robot moves merrily on his preprogrammed way. Line 60: When the last bit of data is reached (RC>8), RC is set equal to 1 and the cycle begins all over again.

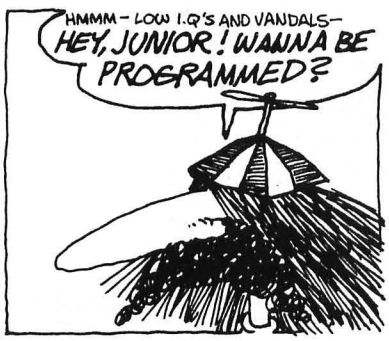
Want to reprogram the Robot? Just add, subtract, or change the coordinates in the DATA statements. Remember, if you change the number of coordinates (delete or add to the DATA statements), you'll have to:

1. Redimension the array in line 120
2. Change the FOR/NEXT loop in line 140 to equal the number of DATA points. For example, if you added one additional set of data, the FOR/NEXT loop would read "FOR A = 1 TO 10."
3. Change the value that follows RC> in line 60 to equal the total number of data points. In the example I've just shown you, the statement would now read "RC>10."

That's all there is to it.

Now You're a Know-it-all!





In addition to your other game design and programming knowledge, you now know how to program a Chaser, Scaredy Cat, Bouncer, and Robot! By combining these four techniques you'll also be able to program Low I.Q.'s and Vandals, but watch out!

With this great store of knowledge tucked neatly under your belt, you can design and program the most sophisticated of arcade games. You can now WOW your friends and neighbors, impress your pets, and amaze yourself!

But to really get their attention, you've got to top your games with wild and crazy sound!

My wife, Petunia, is a wild and crazy gal. I'll ask her if she'd like to sound off.

9
ZOUNDS

Petunia-PETUNIA, DEAR HONEY SWEETYKINS-WHERE ARE YOU?
Bye, SWEETYKINS, GOTTA GO HELP JUNIOR FIND HIS MIND.
AH, IT'S ECSTASY BEING MARRIED TO SUCH A ROMANTIC MAN. BUT NOW IT'S TIME TO TALK ABOUT SOUND.
AND IT'S ESPECIALLY NICE SINCE IT DROWNS OUT MY LOVABLE AND CUTE HUSBAND!

It's about time you *heard* from my gorgeous and talented wife, Petunia. I'll call her.

My Tale of a SOUNDless World

I became frustrated after spending most of my time programming games on really BIG computers. Main frames, supermain frames and the El Biggo Whopper-Doo Mark IV at C.W.I. (Computer Wacko Institute).

I was able to program those biggos in every language known to wackodom. I had them performing games in COBAL, FORTRAN, LISP, even SANSCRIPT! But, none of those gigantic computers could sing, make weird rumbling sounds, meow like a cat, or explode! BAROOOOMMMM! And what, after all, is an arcade game without sound?

I felt that my creativity was being stifled. I realized that I was working in a humdrum environment, a soundless world. I panicked!

My lovable and cute husband and his lovable and cute Atari computer saved me. I now spend my off hours making beautiful, weird, and horrid sounds on his Atari. I love it. It relaxes me. It's therapeutic! My life, and my games, are now full of *SOUND*!

Your Atari can sing, or meow, in as many as four voices. You can use each voice solo, or you can be creative and combine a few, or all four to achieve remarkable sound effects, and some beautiful music.

But the most wonderful thing about the Atari SOUND statement is that it's so easy to use.

A typical SOUND statement looks like this:

SOUND 0,100,10,15

Type this in and press RETURN. When you have heard enough, type END <RETURN>.

Here's what each number in the SOUND statement controls:

VOICE: SOUND 0

Select the voice you'd like to use by assigning a value to the first number following the SOUND command. SOUND 0, SOUND 1, SOUND 2, and SOUND 3 are all available.

PITCH: SOUND 0,100

Select each voice's pitch by setting the second number to a value between 0 and 255. The pitch I've selected in my example is 100.

DISTORTION: SOUND 0,100,10

The third number in a SOUND statement varies distortion, from pure tones to gobbledygook. Examples of pure tones are 10 and 14. Other *even-numbered* values (0, 2, 4, 6, 8, and 12) add different amounts of noise and distortion to your tone. But the number controlling the distortion must be an even value between 0 and 14.

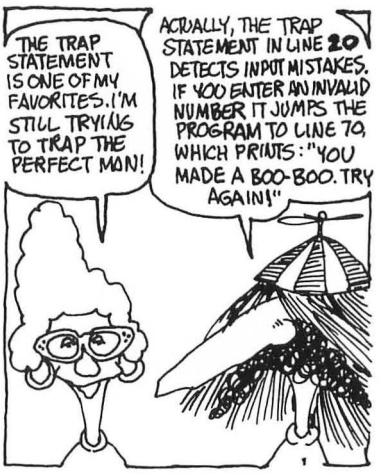
VOLUME: SOUND 0,100,10,15

Vary the loudness of each voice by setting the fourth number to a value between 1 and 15. The value 15 is the LOUDEST, while 1 is just a whisper.

Before I show you how to program BAROOOMs, BONGs, and ZAPs, type in and play around with this short program. It will help you become acquainted with the basic concepts you've just learned.

BASIC Sound

```
10 ?:? "ENTER Pitch, Distortion and Loudness. Press
   RETURN after each entry."
20 TRAP 70:INPUT P,D,L
30 SOUND 0,P,D,L
40 IF PEEK(53279)<>6 THEN GOTO 40
45 . **CHECK OUT SMOKEY PEEKS'S LIST OF GREAT
   PEEKS FOR MORE ON PEEK(53279)**
50 SOUND 0,0,0,0
60 ?:?: "ZOUNDED GREAT!":GOTO 10
70 ? "YOU MADE A BOO-BOO. TRY AGAIN!":GOTO 10
```


THE TRAP STATEMENT IS ONE OF MY FAVORITES. I'M STILL TRYING TO TRAP THE PERFECT MAN!
ACTUALLY, THE TRAP STATEMENT IN LINE 20 DETECTS INPUT MISTAKES. IF YOU ENTER AN INVALID NUMBER IT JUMPS THE PROGRAM TO LINE 70, WHICH PRINTS: "YOU MADE A BOO-BOO. TRY AGAIN!"

A question mark appears on the screen when you RUN this glamorous program. First enter the Pitch, then the Distortion, and finally the Loudness. Hit RETURN after each entry.

Press START when you've heard enough.

I always enjoy playing with combinations of pitch and distortion. Captain Action showed me these interesting combinations—try them, and then create some of your own astounding sounds:

CAR: PITCH = 100, DISTORTION = 4
GENERATOR: PITCH = 100, DISTORTION = 6
ROCKET: PITCH = 100, DISTORTION = 8
AIRPLANE: PITCH = 250, DISTORTION = 12

A Cacophony of Multivoiceferous Zounds

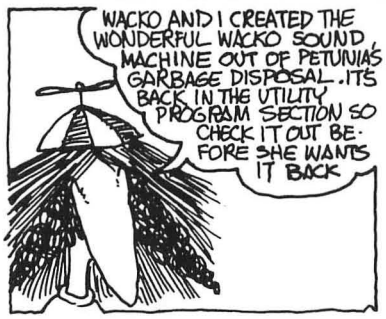
Want to make a cacophony of multivoiceferous zounds? Just replace lines 30 and 50 in the BASIC Sound program with:

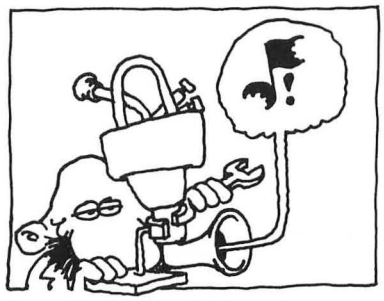
```
30 SOUND 0,P + 4,D,L:SOUND 1,P + 8,D,L:SOUND
   2,P + 12,D,L:SOUND 3,P + 16,D,L

50 FOR OFF = 0 TO 3:SOUND OFF,0,0,0: NEXT OFF
```

Line 50 is a neat routine that turns all four voices off.

See what happens when you use a whole bunch of voices? You get a whole bunch of noise! Change the values that are added to pitch in each SOUND statement. Wilder noise!


WACKO AND I CREATED THE WONDERFUL WACKO SOUND MACHINE OUT OF PETUNIA'S GARBAGE DISPOSAL. IT'S BACK IN THE UTILITY PROGRAM SECTION SO CHECK IT OUT BEFORE SHE WANTS IT BACK



Now That You Understand ZOUND

Now that you understand how the SOUND Statement works, it's time to ask Dr. Wacko to show you how to use SOUND in your arcade games.

Uh-oh, he's still boogalooing to Junior's Purple Smut album.

Well, here are some extra special sound treats for you to enjoy. Spend a little time working through each program while I go pry the great Wacko loose. He shoud be here in a minute. I hope.

Junior's Birthday Present

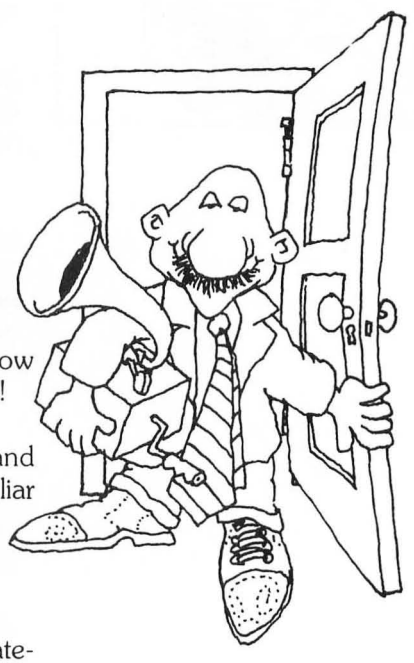
```
10 . JUNIOR'S CHOO-CHOO TRAIN
20 GRAPHICS 17:POKE 712,148:POSITION 1,10:PRINT
   #6;"JUNIOR'S CHOO-CHOO"
30 FOR X=15 TO 0 STEP -1-P:SOUND 1,0,0,X
40 R=INT(RND(0)*300)+1
50 IF R=30 THEN SOUND 3,36,10,10:SOUND
   2,48,10,10:GOSUB 90:SOUND 3,0,0,0:SOUND
   2,0,0,0
60 NEXT X:P=P+0.03
70 IF P>=5 THEN P=5
80 GOTO 30
90 POKE 77,0:POSITION 8,12:PRINT #6;"toot":FOR
   A=1 TO 400:NEXT A:POSITION 8,12:PRINT
   #6;"     ":RETURN
```

A Captain Action Design

```
10 GRAPHICS 17
20 FOR X=10 TO 100:SOUND 0,X,10,10:SOUND
   1,X-2,10,8:SOUND 2,X+2,10,12:NEXT X
30 SOUND 1,0,0,0:SOUND 2,0,0,0
40 POSITION 4,11:PRINT #6;"BAROOOOMMM!"
50 FOR DECAY=15 TO 0 STEP -0.5:SOUND
   0,100,8,DECAY:FOR B=1 TO 20:POKE 712,B:NEXT
   B:NEXT DECAY
60 GRAPHICS 1+32:POKE 712,148
70 POKE 752,1:PRINT "Captain Action designed this
   one!"
80 PRINT :PRINT " Press START to blow up again!"
90 IF PEEK(53279)<>6 THEN GOTO 90
100 GOTO 10
```

Origin Unknown

```
10 DIM WORDS$(20)
20 GRAPHICS 17:POKE 712,15
30 FOR X = 0 TO 2:READ WORDS$,PITCH,
   DISTORT,WAIT,G
40 POSITION 1,10:PRINT #6;WORDS$
50 FOR DECAY = 15 TO 0 STEP  - G
60 SOUND 0,PITCH,DISTORT,DECAY
70 NEXT DECAY
80 POKE 540,WAIT
90 IF PEEK(540)<>0 THEN GOTO 90
95 . Dr. Wacko will explain PEEK(540) when he gets out
   of Junior's room.
100 NEXT X:RESTORE :GOTO 20
110 DATA **SOUND ADVICE**
120 DATA 64,10,60,.1
130 DATA *from mrs. wacko*
140 DATA 120,10,60,1
150 DATA ****burma shave****
160 DATA 150,8,120,.2
```



SOUND Advice from Dr. C. Wacko: "Put SOUND in your arcade games!"

Hey-Yoh! Hey. . . Yoh! Wacko's back! And, I'm going to show you how to add wild and crazy sound to your arcade games!

Remember my great Bong program? Flip back to page 42 and look at the listing. Right side up! Ah-hah! Something very familiar on lines 210 and 260. SOUND Statements!

SOUND and the IF/THEN Statement

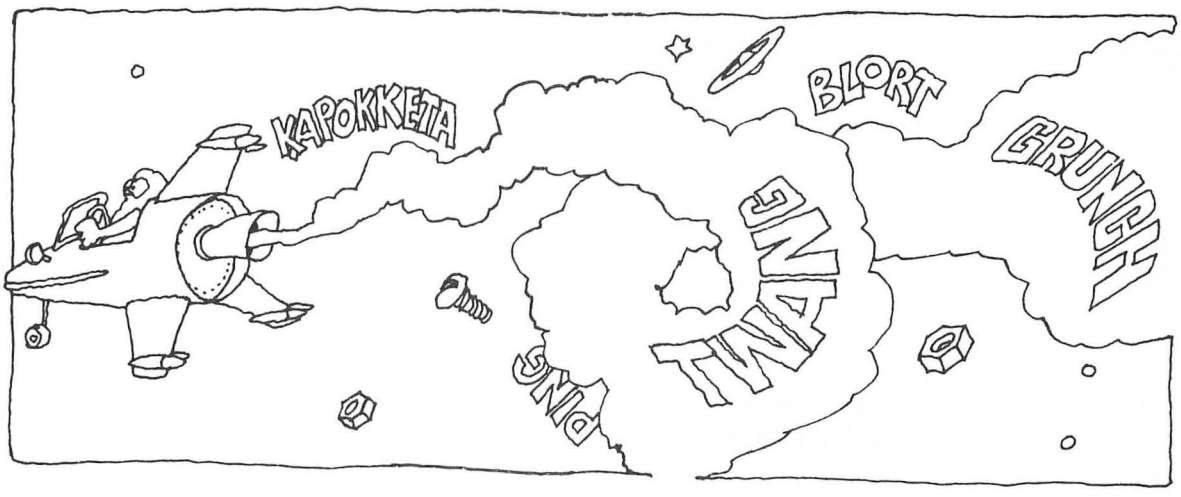
The Bong listing shows a great example of the IF/THEN statement's use in introducing sound into an arcade game. It works like this:

IF, something happens in your game—a ball hits a wall or a rocket ship takes off—

THEN, the SOUND statement is activated.

And glorious sound accompanies and enhances the action!

In the Bong program (line 160), IF the ball hits one of the walls THEN the program jumps to line 210 and SOUND 0,100,10,10 is played. After the color of the walls is changed in line 250, the sound is turned off in line 260 and the program returns back to line 130.



Now that you're an expert sound maker, fool around with the SOUND statment in line 210. Change it to your heart's desire. Captain Action was able to make the computer burp when the ball hit a wall, but you can do better than that! See if you can add more SOUND statements, more than one Voice, to really make the program come alive—PLUNK!

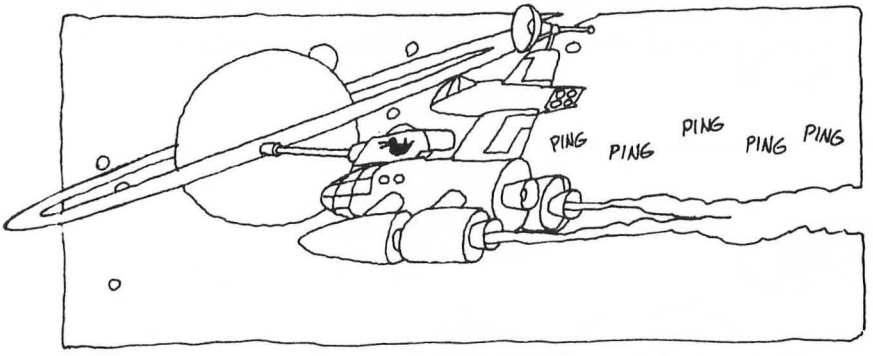
Don't forget to turn off *all* the SOUND Statements in line 260! If you use more than one voice, you might try this FOR/NEXT turns-'em-off trick Mrs. Wacko showed you:

FOR OFF = 0 TO 3:SOUND OFF,0,0,0:NEXT OFF

Decay of the PING

A *ping* sound effect is often used in games to accompany a screen change. For example, in simulating a star cruiser's control console, a ping might sound every time your player changes from battle stations to standby status. A pinging effect is achieved by

decaying the sound: that is, playing the sound a number of times and reducing the volume each play.



Here's how to produce a decaying sound. You'll find many uses for this effect in both games and music.

Ping!

```
10 FOR DECAY = 15 TO 0 STEP - .8:SOUND
   0,60,10,DECAY:NEXT DECAY
```

A decay decreases a note's volume while sustaining its pitch.

Vary the rate of volume decrease by changing the negative value that follows the word STEP in the Ping! program. The smaller the negative number, like − .05, the longer the decay.

Play around with this concept a little. You should be able to get Ping! to sound like a note played on a piano! A decay's opposite number is called an *attack* An attack increases a note's volume while sustaining its pitch. Just change STEP − .8 in the Ping! program to STEP .8 and you'll be attacked!

Although a decay can be used very effectively for music and some special sound effects, it isn't practical to use a decay in the Bong program. If you insert Ping! in place of line 210, the ball will "freeze" against the wall untill the FOR/NEXT loop completes its cycle. Try it and see how the program slows down.

Let's Go to a Ping Subroutine.

In many games the same sound is called for throughout the program. In one of my games the player presses the START key to flip through a variety of screens and maps. A *ping* sound effect

accompanies each screen change. You should use a subroutine to create the *ping*.

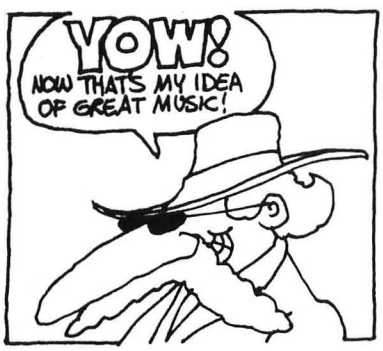
Type in and RUN this little gem to get the idea.

```
  5 . *START key PEEK is on line 10!*
 10 IF PEEK(53279) = 6 THEN GOSUB 100
 20 GOTO 10
 30 . *HERE COMES THE SUBROUTINE!*
100 FOR DECAY = 15 TO 0 STEP -.8:SOUND
    0,60,10,DECAY:NEXT DECAY
200 SOUND 0,0,0,0:RETURN
```

Every time you press the START key, the program branches to the subroutine in line 100. A *ping* sounds, then in line 200 the sound is turned off and the program returns to line 10.

If you are using the same SOUND statement throughout your program, it's easier to GOSUB to a SOUND subroutine. You'll save yourself the effort and time of typing in the same SOUND statements many times, and your program will be more efficient and use less memory.

Changing SOUND with DATA



Complex and changing sounds can be made by reading values into the SOUND statement(s). I'll soon show you how to write some pretty fancy music using this DATA method. First, here's a gruesome example. I've named this short program Mashed Monster. Listen and you'll understand why.

Mashed Monster

```
  0 . *** Read PITCH & DISTORTION values from
    DATA into SOUND Statement ***
 10 FOR X = 0 TO 3:READ P,D
 20 SOUND 0,P,D,12
 25 . *** PAUSE ***
 30 FOR PAUSE = 1 TO 200:NEXT PAUSE
 40 NEXT X:RESTORE:GOTO 10
 50 . *** The DATA is in pairs: (PITCH,DISTORTION)
    ***
100 DATA 60,2,85,10,150,6,100,8
```

Even though Snidely's wild about my Mashed Monster program, most of the people who play your arcade games will have a more refined musical sense. (No offense to your sense intended, Snidely.) They deserve to hear the real McCoy. Music with a beat. Music they can relate to. Music that sounds like *music*!

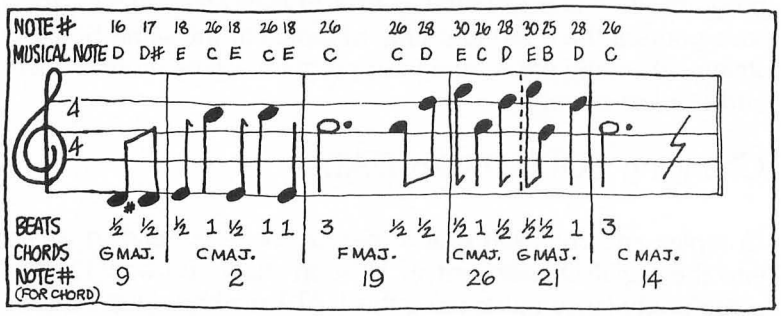
See, even C.A. relates to my message!

Your Game Comes Alive to the SOUND of Music

During the introduction, while the name of your great game is flashing in brilliant lights on the silver screen, you may want to add a touch of professionalism, a little music. You may also want to add music to your game to reward the player when he or she reaches a new level of play, or wins.

A NOTE IS A MUSICAL SOUND AND A CHORD IS A SERIES OF TONES BLENDING TOGETHER, QUITE UNLIKE PETUNIA'S KITTY CHORALE.
SEE THE NOTE IN MY CHORD?

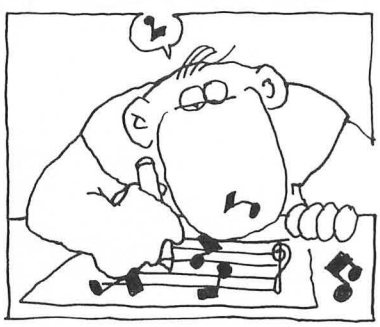
Before you can add music to your game, you'll need to take a look at some real music first. Musical notation looks like this:


NOTE # 16 17 18 26 18 26 18 26 26 28 30 26 28 30 25 28 26
MUSICAL NOTE D D# E C E C E C C D E C D E B D C
BEATS ½ ½ ½ 1 ½ 1 1 3 ½ ½ ½ 1 ½ ½½ 1 3
CHORDS G MAJ. C MAJ. F MAJ. C MAJ. G MAJ. C MAJ.
NOTE # (FOR CHORD) 9 2 19 26 21 14

But you really don't have to know much about music to convert this short bit of notation into melodious Atari computer music. I've done most of the work for you!

I've listed the name of each note above its symbol, indicated the note's duration below it, and shown the chord that accompanies each group of notes.

Scribble on Your Sheet Music!



The first step in adapting music to your Atari is to take the sheet music and mark it up like I've done. I didn't know enough about music to do it on my own, so I had to ask Petunia to help me. You might ask a friend or music instructor to help you mark up your sheet music. Or, if you're familiar with music, or really adventurous, you might want to go ahead and compose your own music.

Now that you've scribbled all over the sheet music, you're ready to play music on your Atari computer.

A Wacked-Out Musical Chart

Here's a handy chart to help you convert musical notes into SOUND statements.

Listed in the PITCH column are the numbers that will produce each MUSICAL NOTE when plugged into a SOUND statement.

Because music is played using pure tones, distortion value 10 is always used in the SOUND statement. To play a middle C your SOUND statement will look like this:

SOUND 0,60,10,15

To hear what a G note sounds like, for example, just substitute the number 81 for 60.

NOTE	PITCH	MUSICAL NOTE
1	255	B
2	243	C
3	230	C# or Db
4	217	D
5	204	D# or Eb
6	193	E
7	182	F
8	173	F# or Gb
9	162	G
10	153	G# or Ab
11	144	A
12	136	A# or Bb
13	128	B
14	121	C
15	114	C# or Db
16	108	D
17	102	D# or Eb
18	96	E
19	91	F
20	85	F# or Gb
21	81	G
22	76	G# or Ab
23	72	A

24	68	A# or Bb
25	64	B
26	60	C
27	57	C# or Db
28	53	D
29	50	D# or Eb
30	47	E
31	45	F
32	42	F# or Gb
33	40	G
34	37	G# or Ab
35	35	A
36	33	A# or Bb
37	31	B
38	29	C
39	27	C# or Db
40	26	D
41	24	D# or Eb
42	23	E
43	22	F
44	21	F# or Gb
45	19	G
46	18	G# or Ab
47	17	A
48	16	A# or Bb
49	15	B
50	14	C

In this chart, note 1 is a low B, and note 50 is a high C. I've assigned the numbers 1 through 50 to the pitch values of each note so you'll be able to produce chords by using my patented "Add-a-Chord" method. I'll demonstrate this soon.

Wacko's Musical Recipe

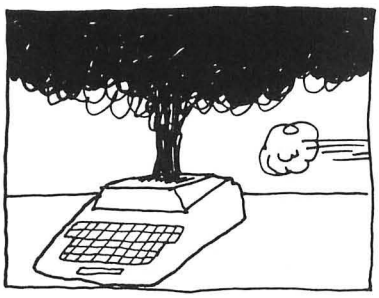
Creating beautiful music is like preparing a gourmet meal. A pinch of this and a sprinkle of that, all combined in the correct proportions and, *voila*, a masterpiece!

"The Sting," with Tofu Pasta and Sauce (makes one serving)

You'll need:

1. Musical notes placed in a lightly oiled array

2. A routine to play individual notes
3. A routine to play chords
4. A routine to time each note's duration



Combine all ingredients in your Atari computer, stir well, and simmer a few minutes. That's all there is to it! Let's take it step-by-step:

1. Place notes into an array.

First we'll convert my Wacked-Out Musical Chart into an array. This will assign NOTES 1 through 50 to PITCHES 255 to 14. Here's the short routine that'll do this. We'll add this to the other ingredients as we cook up our tune, so use the line numbers I've assigned.

Type in this short routine. Don't forget to stir well!

PITCH Array Routine

```
60 DIM N(50):. *** DIMENSION ARRAY ***
70 FOR NBR=1 TO 50:READ
   PITCH:N(NBR)=PITCH:NEXT NBR
80 . *** PITCH ARRAY DATA ****
90 DATA 255, 243, 230, 217, 204, 193, 182, 173, 162,
   153, 144, 136, 128, 121, 114, 108, 102, 96, 91, 85,
   81, 76, 72, 68, 64
100 DATA 60, 57, 53, 50, 47, 45, 42, 40, 37, 35, 33, 31,
   29, 27, 26, 24, 23, 22, 21, 19, 18, 17, 16, 15, 14
```

If you'd like to see how this routine works, and what it does, just add this line to the PITCH Array Routine, and RUN it!

```
75 FOR X=1 TO 50:PRINT " N(";X;") = ";N(X):NEXT
   X:STOP
```

Voila! You've just produced a copy of my Wacked-Out Musical Chart on your computer!

So you won't have to retype this short routine, remove line 45, then LIST it to your disk or cassette like this:

Disk owners: LIST"D:ARRAY" <RETURN>

Cassette owners: LIST"C:" <RETURN>

2. A routine to play individual notes

This one's really simple:

SOUND 0,N(P),10,14

Each individual note will be played using voice 0, have a PITCH selected from the array you've just created (N(P)), be a pure tone (10), and have a loudness of 14.

3. A Routine to Play Chords

This is a two-line routine. Here's the first line:

```
10 P0 = N(P):P1 = N(CRD):P2 = N(CRD + 4):
   P3 = N(CRD + 7)
```

The first statement, P0 = N(P), is used to produce a single note in SOUND 0.

The second statement, P1 = N(CRD), is used in SOUND 1, to produce the beginning note of each chord.

In statements three and four harmony is achieved by adding 4 and 7 to the beginning chord note.

Here's the second line of this routine to show you how it all fits together:

```
20 SOUND 0,P0,10,6:SOUND 1,P1,10,4:SOUND
   2,P2,10,4:SOUND 3,P3,10,4
```

Now we'll combine these two ingredients, add a few spices, and I'll show you how chords are produced.

Chords!

```
 5 GOTO 60
10 P0 = N(P):P1 = N(CRD):P2 = N(CRD + 4)
   :P3 = N(CRD + 7)
20 SOUND 0,P0,10,6:SOUND 1,P1,10,4:SOUND
   2,P2,10,4:SOUND 3,P3,10,4
30 FOR PAUSE = 1 TO 500: NEXT PAUSE
40 IF CRD = 14 AND P = 26 THEN FOR PAUSE = 0 TO
   500:NEXT PAUSE:FOR OFF = 0 TO 3:SOUND
```

```
    OFF,0,0,0:NEXT OFF:RESTORE 200
 50 GOTO 200
 60 DIM N(50):. *** DIMENSION ARRAY ***
 70 FOR NBR = 1 TO 50:READ
    PITCH:N(NBR) = PITCH:NEXT NBR
 80 . **** PITCH ARRAY DATA ****
 90 DATA 255, 243, 230, 217, 204, 193, 182, 173, 162,
    153, 144, 136, 128, 121, 114, 108, 102, 96, 91, 85,
    81, 76, 72, 68, 64
100 DATA 60, 57, 53, 50, 47, 45, 42, 40, 37, 35, 33, 31,
    29, 27, 26, 24, 23, 22, 21, 19, 18, 17, 16, 15, 14
190 . *** CORD & NOTE DATA ***
200 READ CRD,P:GOTO 10
210 DATA 2, 14
220 DATA 4, 16
230 DATA 6, 18
240 DATA 7, 19
250 DATA 9, 21
260 DATA 11, 23
270 DATA 13, 25
280 DATA 14, 26
```

RUN this Chords program; it continuously plays chords from low to middle C.

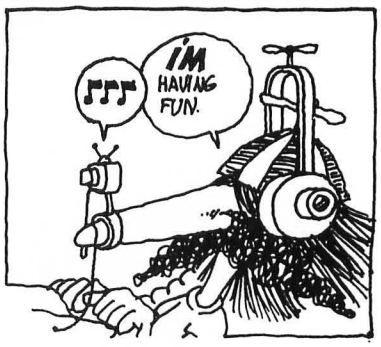
I've added new lines 5, 30 through 50, and most important the Chord & Note Data in lines 200 through 280 so you'll be able to hear chords played on your Atari computer.

We'll use all of these concepts in the final composition. Look at my Wacked-Out Music Chart and I'll show you how the Chords program works. Beginning in line 60, the program places notes into an array. Then, in line 200, the first line of data—210 DATA 2, 14—is read. The number 2 represents PITCH 243, the musical note low C, the number 14 represents PITCH 121, middle C.

After reading this line of data, the program goes to line 10 where the variables CRD and P are replaced with the values just read in line 210.

The chord is played in line 20, pauses for a few seconds in line 30, then in line 50 goes back to line 200 for the next chord. Line 40 waits for the DATA values to reach 14 and 26. When they do, the last note is held a little longer then the rest; all four voices are

turned off; the DATA is restored; and the chords are played once again, beginning with low C.



Changing the DATA in lines 210 through 280 can produce some very laid-back effects!

4. The final ingredient: a routine to time each note's duration

Now that your kitchen is a mess, its time to add the final ingredient.

Ms. Peeky and Slow Poke discovered PEEK(540) and POKE 540, the final ingredient. When you POKE a number into location 540, your Atari will count down from that number in 1/60th-second steps until it reaches 0. These means that if you POKE 540 with 60 (POKE 540,60), your Atari will count down to 0 in 1 second.

POKE 540 will become the main ingredient of a routine to time each note's duration.

Here's a short example that illustrates this mystical phenomena.

```
 0 REM *** MYSTICAL TIMING ROUTINE ***
10 INPUT WAIT
20 IF WAIT>255 OR WAIT<0 THEN WAIT = 0:GOTO 10
30 SOUND 0,60,10,14
40 POKE 540,WAIT
50 IF PEEK(540)<>0 THEN GOTO 50
60 SOUND 0,0,0,0
70 GOTO 10
```

First RUN this routine. Next, enter a number between 1 and 255 when you see the prompt(?), then press RETURN. The higher the value you enter, the longer the note's duration.

NOTE	DATA VALUE	DURATION
𝅝	= 80	= 4 BEATS
𝅗𝅥.	= 60	= 3 BEATS
𝅗𝅥	= 40	= 2 BEATS
♩	= 20	= 1 BEAT
♪	= 10	= ½ BEAT

Lines 40 and 50 are the heart of this short routine. Line 40 counts down from the number you've entered, WAIT to 0. Line 50 won't allow the program to go to line 60, which turns off the sound, until the value in location 540 equals 0.

The Value of WAIT

In most cases, unless the music doesn't sound right to me, I use these values in DATA statements to time the duration of each note.

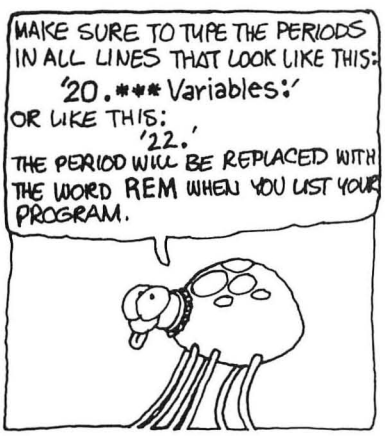
Combine All Ingredients and Simmer

Hold on to your chef's hat. This is the BIGGIE!

Here's the *complete* program listing, the one you've been waiting for. This concoction includes the four ingredients we've discussed:

1. Musical notes placed in a lightly oiled array.
2. One routine that plays individual notes.
3. One routine that plays chords.
4. One routine that times each note's duration.

Plus, a few spices and herbs.


MAKE SURE TO TYPE THE PERIODS IN ALL LINES THAT LOOK LIKE THIS:
'20.*** Variables:'
OR LIKE THIS:
'22.'
THE PERIOD WILL BE REPLACED WITH THE WORD REM WHEN YOU LIST YOUR PROGRAM.

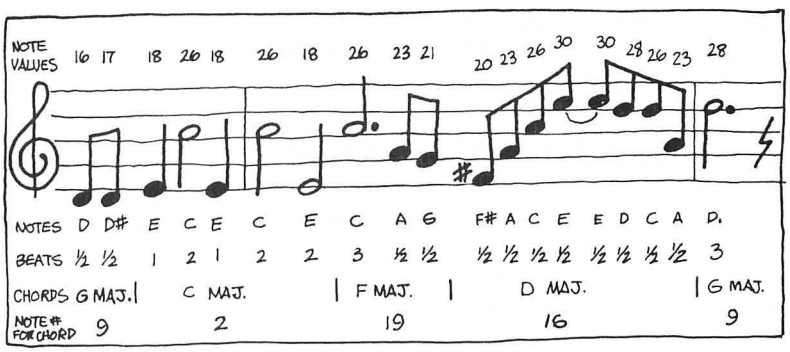
Presenting! A Wacko Masterpiece: "The Sting," with Tofu Pasta and Sauce

```
10 GRAPHICS 1:POKE 712,148:POSITION 5,10:PRINT
   #6;"THE STING"
20 . *** Variables: P = Pitch - N = Note - CRD = Cord
   -WAIT = Note's duration ***
22 .
24 .
30 DIM N(50):. *** DIMENSION ARRAY **
32 .
34 .
40 . *** 1. PLACE NOTES IN ARRAY ***
```

```
 50 FOR NBR = 1 TO 50:READ
    PITCH:N(NBR) = PITCH:NEXT NBR:GOTO 180
 52 .
 54 .
 60 . *** 2. ROUTINE THAT PLAYS INDIVIDUAL NOTES
    ***
 70 SOUND 0,N(P),10,14:GOTO 120
 72 .
 74 .
 80 . *** 3. ROUTINE THAT PLAYS CORDS ***
 90 P0 = N(P):P1 = N(CRD):P2 = N(CRD + 4):P3 = N(CRD + 7)
100 SOUND 0,P0,10,6:SOUND 1,P1,10,4:SOUND
    2,P2,10,4:SOUND 3,P3,10,4
102 .
104 .
110 . *** 4. ROUTINE TO TIME EACH NOTE'S
    DURATION ***
120 POKE 540,WAIT
130 IF PEEK(540)<>0 THEN POKE 77,0:GOTO 130
140 SOUND 0,0,0,0:RETURN
142 .
144 .
150 . *** END OF MUSIC - PRESS START
    ROUTINE ***
160 FOR OFF = 0 TO 3:SOUND OFF,0,0,0:NEXT
    OFF:POKE 752,1:PRINT CHR$(125):PRINT
    " PRESS START TO REPLAY"
170 RESTORE 250:IF PEEK(53279)<>6 THEN GOTO 170
180 GOTO 240
182 .
184 .
190 . *** INITIALIZING DATA ****
200 DATA 255, 243, 230, 217, 204, 193, 182, 173, 162,
    153, 144, 136, 128, 121, 114, 108, 102, 96, 91, 85, 81,
    76, 72, 68, 64
210 DATA 60, 57, 53, 50, 47, 45, 42, 40, 37, 35, 33, 31,
    29, 27, 26, 24, 23, 22, 21, 19, 18, 17, 16, 15, 14 212.
214 .
220 . *** READ & PLAY CORDS, PITCH AND WAIT
    DATA ***
230 .
240 READ CRD, P,WAIT:GOSUB 90:READ
    P,WAIT:GOSUB 70
250 DATA 9, 16, 10, 17, 10
260 .
```

```
270 READ CRD,P,WAIT:GOSUB 90:FOR X=1 TO
    4:READ P,WAIT:GOSUB 70:NEXT X
280 DATA 2, 18, 10, 26, 23, 18, 10, 26, 31, 18, 20
290 .
300 READ CRD,P,WAIT:GOSUB 90:FOR X=1 TO
    2:READ P,WAIT:GOSUB 70:NEXT X
310 DATA 19, 26, 60, 26, 10, 28, 10
320 .
330 READ CRD,P,WAIT:GOSUB 90:FOR X=1 TO
    2:READ P,WAIT:GOSUB 70:NEXT X
340 DATA 26, 30, 10, 26, 20, 28, 10
350 .
360 READ CRD,P,WAIT:GOSUB 90:FOR X=1 TO
    2:READ P,WAIT:GOSUB 70:NEXT X
370 DATA 21, 30, 10, 25, 10, 28, 20
380 .
390 READ CRD,P,WAIT:GOSUB 90:FOR X=1 TO
    2:READ P,WAIT:GOSUB 70:NEXT X
400 DATA 14, 26, 20, 9, 20, 2, 40
410 .
420 GOTO 160
```

Ready to add the next stanza of this great classic? Here's the musical notation.


NOTE VALUES 16 17 18 26 18 26 18 26 23 21 20 23 26 30 30 28 26 23 28
NOTES D D# E C E C E C A G F# A C E E D C A D
BEATS ½ ½ 1 2 1 2 2 3 ½ ½ ½ ½ ½ ½ ½ ½ ½ ½ ½ 3
CHORDS G MAJ. | C MAJ. | F MAJ. | D MAJ. | G MAJ.
NOTE # FOR CHORD 9 2 19 16 9

To add the first chord and two notes of the second stanza just add new lines 420, 430 and 440 as I've done below:

```
420 READ CRD,P,WAIT:GOSUB 90:READ
    P,WAIT:GOSUB 70:GOTO 160

430 DATA 9, 16, 10, 17, 10

440 GOTO 160
```

The chord and two notes represented in new lines 420 and 430 are the same notes that began the first stanza of "The Sting."

The first statement in line 420 READs the first three numbers of DATA in line 430: 9, 16, and 10. The number 9 is the note number that makes up the G major chord (look at the Wacked-Out Music Chart). The number 16 is the first note in the stanza, D. The number 10 is used to time the duration of the D note.

After READing these three values, the program GOSUBs to line 90, where the G major chord and single D note are played. When PEEK(540) on line 130 reaches 0, the sound is turned off in line 140 and the program returns to the second statement in line 420.

The second statement in line 420 READs the last two numbers of DATA in line 430: 17 and 10. The number 17 is the second note of the stanza, D#. The number 10 is used to time the duration of this note.

After READing these two notes, the program GOSUBs to line 70 where the single D# note is played. After the SOUND statement on line 70 is turned on, the program jumps to the timing routine on line 120, and when PEEK(540) reaches 0, the sound is turned off in line 140. The program returns to line 430 which sends it on its way to the END OF MUSIC - PRESS START ROUTINE on line 160. WHEEEW!

Cook Up a Tune of Your Own!

I've given you the basic ingredients used to cook up great music. But before you cook up a tune of your own, study the way the entire recipe goes together by stepping through the flow of this program as I've done above.

Bon appetit!

A Bonus Balonous Section

And now, by special request from Captain Action, I've added a Bonus Balonous Section to this glorious book. In it you'll learn all the mysteries of Player-Missile graphics. Then you'll be treated to the bonus arcade game Myrtle the Turtle!

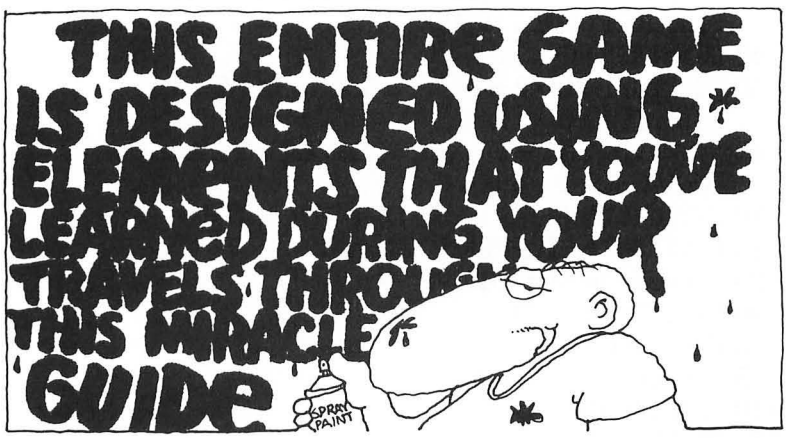
These and more exciting adventures are awaiting you when you turn the page and wander through my bogus Bonus Section.

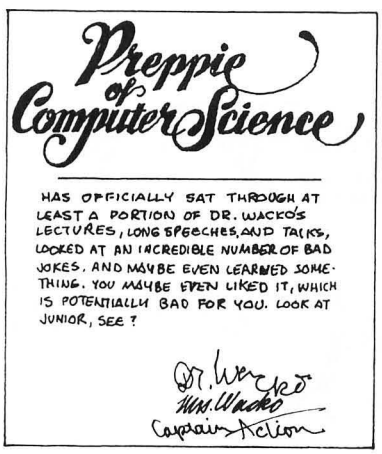
STOP . . . HALT . . . ARRETEZ-VOUS!

Before you start wandering around aimlessly, I'd like to ask you to STOP!

Did you stop? Thanks.

Here's the most EXCITING program that's been presented in this book so far! It's called Shootout. Yep, you guessed it, it's a two-player arcade game. Two players, each with his or her own joystick, maneuver two animated characters. You fire at your opponent by pointing the joystick in the other guy's direction while pressing the trigger. The first player to hit the other fifteen times wins the shootout!




Preppie of Computer Science
HAS OFFICIALLY SAT THROUGH AT LEAST A PORTION OF DR. WACKO'S LECTURES, LONG SPEECHES, AND TALKS, LOOKED AT AN INCREDIBLE NUMBER OF BAD JOKES, AND MAYBE EVEN LEARNED SOMETHING. YOU MAYBE EVEN LIKED IT, WHICH IS POTENTIALLY BAD FOR YOU. LOOK AT JUNIOR, SEE?
Dr. Wacko
Wm. Wacko
Captain Action

To help refresh your memory about the wondrous elements that make up this game I've added helpful REM statements throughout the program. Plus drawings of the redefined characters and a complete description of the shoot sequence! Flip back to previous chapters as you work through this program and review any material that might be a little fuzzy.

Preppie of Computer Wacko Science

Once you're unfuzzed, fill in this honorary certificate and award yourself the "Preppie of Computer Wacko Science." It's time to party! Congratulations!

Delve into this program, improve it, make it worse, change the sound, the colors, the characters. Design your own game! You can do it! YOU'RE A PREPPIE OF COMPUTER WACKO SCIENCE!

The H.D.C.W.S.

Want to get your Humongous Degree of Computer Wacko Science? After you've mastered Shootout, move on to the Bonus Section! Congratulations again!

Shootout

```
 10 DIM X(15),Y(15)
 20 GOSUB 770
 25 GRAPHICS 18:POKE 712,45:GOTO 670
 30 GRAPHICS 18:POKE 756,ST/256:POKE
    708,30:POKE 709,92:POKE 711,158:POKE
    710,205:POKE 712,128
 40 GOSUB 690
 50 X1 = 0:Y1 = 0:X2 = 19:Y2 = 5:Z1 = 32:Z2 = 32:
    S1 = 0:S2 = 0
 60 XB1 = X1:YB1 = Y1:XB2 = X2:YB2 = Y2:A = 0:
    B = 0:C = 0:D = 0:E = 0
 70 COLOR 32:D = D + 1:IF D>1 THEN D = 0
 80 PLOT XB1,YB1:PLOT XB2,YB2
 85 . Weird character 1
 90 COLOR 166 + D
100 PLOT X1,Y1
105 . Weird character 2
110 COLOR 136 + D
120 PLOT X2,Y2
130 GOSUB 310
135 . If one of the two weirdos is hit, the program goes to
    line 470 and explosion takes place.
140 IF AISD = 1 THEN X3 = X1:Y3 = Y1:S2 = S2 + 1:GOTO
    470
150 IF BISD = 1 THEN X3 = X2:Y3 = Y2:S1 = S1 + 1:GOTO
    470
155 .
160 XB1 = X1:YB1 = Y1:XB2 = X2:YB2 = Y2
170 A = STICK(0):B = STICK(1):IF A<>15 THEN POKE
    77,0
180 F = F + 1:IF F>3 THEN G = G + 1:F = 0:IF G>3 THEN
    G = 0
```

```
190 SOUND 0,G*5 - F + 60,10,F + G:SOUND
    1,G*4 - F*4 + D + 55,10,D*4 + 1
200 X1 = X1 + X(A):X2 = X2 + X(B)
210 Y1 = Y1 + Y(A):Y2 = Y2 + Y(B)
220 IF X1<0 OR X1>19 THEN X1 = 19 - XB1
230 IF Y1<0 OR Y1>11 THEN Y1 = 11 - YB1
240 IF X2<0 OR X2>19 THEN X2 = 19 - XB2
250 IF Y2<0 OR Y2>11 THEN Y2 = 11 - YB2
255 . Keep two weirdos within boundaries of maze.
260 LOCATE X1,Y1,Z1:LOCATE X2,Y2,Z2
270 IF Z1<>32 THEN X1 = XB1:Y1 = YB1
280 IF Z2<>32 THEN X2 = XB2:Y2 = YB2
285 .
290 SOUND 0,0,0,0:SOUND 1,0,0,0
300 GOTO 70
310 AISD = 0:BISD = 0:A = STRIG(0):B = STRIG(1):IF A = 0
    THEN GOSUB 370:IF BISD = 1 THEN RETURN
315 . Weirdo 2 shoots
320 IF B = 0 THEN GOSUB 420
325 .
330 RETURN
340 AISD = 0:BISD = 0:A = STRIG(0):B = STRIG(1):IF B = 0
    THEN GOSUB 420:IF AISD = 1 THEN RETURN
345 . Weirdo 1 shoots
350 IF A = 0 THEN GOSUB 370
355 .
360 RETURN
365 . You've got to point in direction of fire or RETURN to
    350
370 F = STICK(0):IF F = 15 THEN RETURN
375 .
380 FOR A = 1 TO 4:X3 = X1 + X(F)*A:Y3 = Y1 + Y(F)*A:IF
    X3<0 OR Y3<0 OR X3>19 OR Y3>11 THEN 410
390 LOCATE X3,Y3,Z3:COLOR 37:PLOT X3,Y3:SOUND
    0,X3 + Y3 + 20 + A*2,10,10:COLOR Z3:SOUND
    0,0,0,0:PLOT X3,Y3
400 IF Z3 = 136 + D THEN BISD = 1:POP :RETURN
410 NEXT A:RETURN
415 . You've got to point in direction of fire or RETURN to
    350
420 F = STICK(1):IF F = 15 THEN RETURN
425 .
430 FOR A = 1 TO 4:X3 = X2 + X(F)*A:Y3 = Y2 + Y(F)*A:IF
    X3<0 OR Y3<0 OR X3>19 OR Y3>11 THEN 410
```

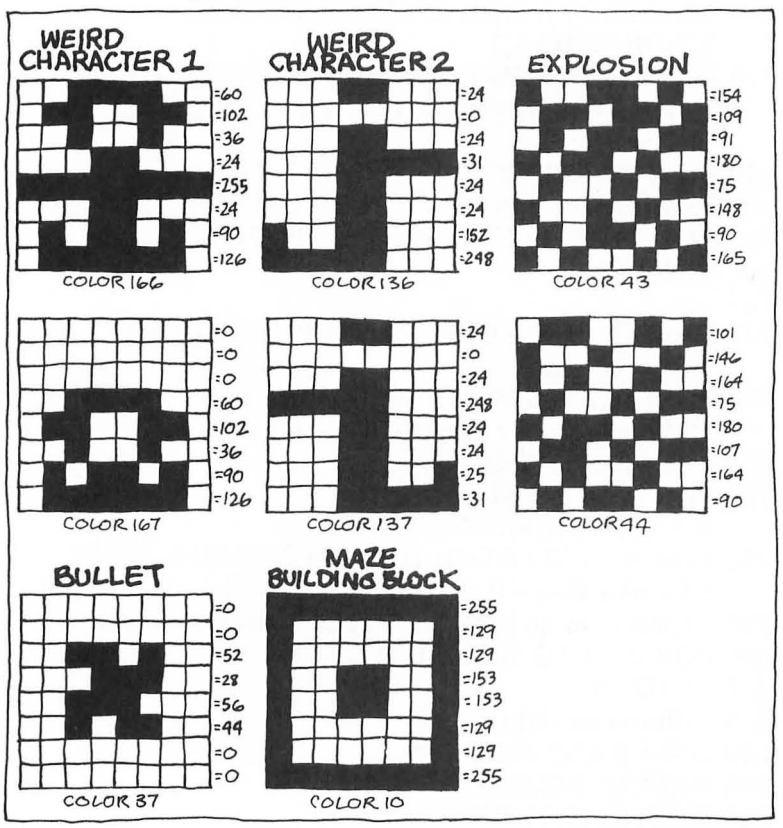
```
440 LOCATE X3,Y3,Z3:COLOR 37:PLOT X3,Y3:SOUND
    0,X3 + Y3 + 10 + A*3,10,10:COLOR Z3:SOUND
    0,0,0,0:PLOT X3,Y3
450 IF Z3 = 166 + D THEN AISD = 1:POP :RETURN
460 NEXT A:RETURN
465 . One of the weirdos has been hit! Explosion sequence.
470 FOR A = 1 TO 3:FOR B = 1 TO 2:FOR Z3 = 0 TO
    1:SOUND 0,Z3*10 + B*5 + A*20,
    12,A*2 + B + Z3:COLOR 43 + Z3:PLOT X3,Y3
480 NEXT Z3:NEXT B:NEXT A:COLOR 32:PLOT
    X3,Y3:SOUND 0,0,0,0
490 COLOR 32:PLOT XB1,YB1:PLOT XB2,YB2:PLOT
    X1,Y1:PLOT X2,Y2
495 . Randomly position weirdo at a new screen location.
500 A = INT(RND(0)*9 + 1):ON A GOSUB
    520,530,540,550,560,570,580,590,600
510 GOTO 610
520 X3 = 0:Y3 = 0:RETURN
530 X3 = 9:Y3 = 0:RETURN
540 X3 = 19:Y3 = 0:RETURN
550 X3 = 0:Y3 = 6:RETURN
560 X3 = 19:Y3 = 5:RETURN
570 X3 = 0:Y3 = 11:RETURN
580 X3 = 10:Y3 = 11:RETURN
590 X3 = 19:Y3 = 11:RETURN
600 X3 = 9:Y3 = 5:RETURN
605 .
610 IF AISD = 1 THEN AISD = 0:X1 = X3:Y1 = Y3
620 IF BISD = 1 THEN BISD = 0:X2 = X3:Y2 = Y3
625 . Print the score.
630 POSITION 4,0:PRINT #6;S1;:POSITION 14,0:PRINT
    #6;S2;
635 . Winning sequence.
640 IF S1>14 OR S2>14 THEN FOR A = 0 TO 3:SOUND
    A,0,0,0:NEXT A:GOTO 670
645 . Color and sound accompany each score.
650 FOR A = 0 TO 6:FOR B = 0 TO 4:SOUND
    0,77 - A*10 - B*4,10,10:POKE
    712,77 - A*10 - B*4:NEXT B:NEXT A
660 SOUND 0,100,10,15:POKE 712,15:FOR A = 0 TO
    10:NEXT A:SOUND 0,0,0,0:POKE 712,128:GOTO 70
670 IF PEEK(53279)<>6 THEN POSITION 5,5:PRINT
    #6;"PRESS START":POKE 709,PEEK(20):GOTO 670
675 . Begin game.
680 GOTO 30
```

```
685 . Draw the playing field.
690 COLOR 10:PLOT 2,0:DRAWTO 8,0:PLOT
    11,0:DRAWTO 17,0:PLOT 2,11:DRAWTO 8,11:PLOT
    11,11:DRAWTO 17,11
700 PLOT 2,2:DRAWTO 0,2:DRAWTO 0,4:PLOT
    0,7:DRAWTO 0,9:DRAWTO 2,9
710 PLOT 17,2:DRAWTO 19,2:DRAWTO 19,4:PLOT
    19,7:DRAWTO 19,9:DRAWTO 17,9
720 PLOT 4,1:DRAWTO 4,10:PLOT 15,1:DRAWTO
    15,10:PLOT 5,4:DRAWTO 7,4:DRAWTO
    7,7:DRAWTO 5,7
730 PLOT 14,4:DRAWTO 12,4:DRAWTO 12,7:DRAWTO
    14,7:PLOT 2,4:DRAWTO 2,7:PLOT 17,4:DRAWTO
    17,7
740 PLOT 7,2:DRAWTO 12,2:PLOT 7,9:DRAWTO
    12,9:PLOT 9,3:PLOT 9,4:PLOT 10,7:PLOT 10,8
750 COLOR 32:PLOT 4,5:PLOT 7,6:PLOT 12,5:PLOT 15,6
60  RETURN
765 .
770 GRAPHICS 18:POKE 712,148:POSITION 6,4:PRINT
    #6;"shoot out"
775 POSITION 9,5:PRINT #6;"is":POSITION 7,6:PRINT
    #6;"loading":ST = (PEEK(742) - 2)*256
778 . Character redefinition
780 FOR A = 0 TO 511:POKE
    A + ST,PEEK(57344 + A):NEXT A
790 FOR A = 1 TO 8:FOR B = 0 TO 7:READ C:POKE
    ST + (A + 4)*8 + B,C:NEXT B:NEXT A
795 . Data in to an array to control bullet's direction.
800 FOR A = 1 TO 15:READ B,C:X(A) = B:Y(A) = C:NEXT A
810 RETURN
815 . Character data.
820 DATA 0,0,52,28,56,44,0,0
830 DATA 60,102,36,24,255,24,90,126
840 DATA 0,0,0,60,102,36,90,126
850 DATA 24,0,24,31,24,24,152,248
860 DATA 24,0,24,248,24,24,25,31
870 DATA 255,129,129,153,153,129,129,255
880 DATA 154,109,91,180,75,148,91,165
890 DATA 101,146,164,75,180,107,164,90
895 . Bullet's direction of movement data.
900 DATA 0,0,0,0,0,0,0,0,1,1,1, - 1,1,0,0,0, - 1,
    1, - 1, - 1, - 1,0,0,0,0,1,0, - 1,0,0
```

Important programming note: Enter all Underlined words between quotation marks as inverse characters. Press the Atari symbol key to do this. One example of this usage is PRESS START in line 670.

Shootout Explained

Here are the characters used in Shoot-Out. Their data is listed in lines 820 through 890.



The two weirdos are animated in lines 70 through 130. Weird character 1 is either COLOR 166 or 167. Weird character 2 is printed as either COLOR 136 or 137.

COLOR 10 is used in lines 690 through 750 to draw the maze.

COLOR 136 prints the bullet in lines 390 and 440.

The explosion occurs in line 470 flipping between COLOR 43 and 44.

To look at each redefined character, RUN Shootout, press BREAK and enter this code in the immediate mode:

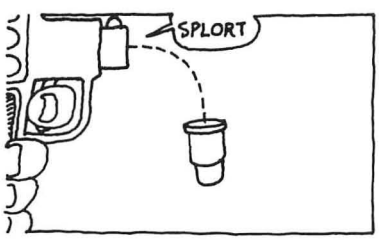
GRAPHICS 2:POKE 756,ST/256:POSITION 5,5:PRINT CHR$(136) <RETURN>.

To see any character displayed, place its ATASCII value in the parentheses after CHR$.

To speed things up, and flip through the entire character set, RUN Shootout, then press break GOTO 1000, this one line program:

1000 GRAPHICS 2:POKE 756,ST/256:FOR A = 0 TO 255:POSITION 5,5:PRINT CHR$(A):FOR PAUSE = 1 TO 200:NEXT PAUSE:NEXT A:STOP

The Shoot'em Sequence



The data in Line 900, loaded into the X and Y arrays in the FOR/NEXT loops in line 800, is cleverly used to send the bullet flying in the direction that the joystick is pointed. Here's how this works:

First two arrays (X and Y) are loaded with data from line 900. Here's what the arrays look like, and what they do:

X(0), Y(0):	Not used
X(1), Y(0):	Not used
X(2), Y(0):	Not used
X(3), Y(0):	Not used
X(4), Y(0):	Not used
X(5) = 1, Y(5) = 1:	DOWN RIGHT
X(6) = 1, Y(6) = − 1:	UP RIGHT
X(7) = 1, Y(7) = 0:	RIGHT
X(8), Y(8):	Not used
X(9) = − 1, Y(9) = 1:	DOWN LEFT
X(10) = − 1, Y(10) = − 1:	UP LEFT
X(11) = − 1, Y(11) = 0:	LEFT
X(12), Y(12):	Not used
X(13) = 0, Y(13) = 1:	DOWN
X(14) = 0, Y(14) = − 1:	UP
X(15), Y(15):	Not used

Turn back to Chapter 6, page 100. It shows the numbers generated when the joystick is pointed in each of its eight possible directions. Now look at lines 370 and 380 of Shootout. In line 370 F equals STICK(0) and in line 380 the bullet's direction of movement, X3, is determined by the the numbers returned in X(F) and Y(F)—the arrays we've just set up. Pretty sneaky!

Line 380 also determines how far the bullet will travel. I have it set for 4 spaces (A = 1 to 4). To see what happens when the bullet travels further, change this statement to A = 1 to 7.

Now that you have a grasp of the fundamentals, here's a step by step, quicky tour of the shootem' up sequence:

Line 310: If weird character 1 fires a bullet the program goes to line 370.

Line 370: If the player is not pointing the joystick, the program RETURNs back to line 310.

Line 380: The bullet travels in the direction that the joystick is pointed.

Lines 390 and 400: A LOCATE statement is used to see if weird character 2 has been hit by the bullet. Z3 is the bullet's location and COLOR 136+D is weird character 2. In line 400, IF Z3 = 136 + 6 THEN BISD = 1. (BISD stands for "B" Is Dead. AISD stands for "A" Is Dead) The POP makes the program skip past the last RETURN at line 360 and move up to line 140.

In line 150, since BISD now equals 1, the bullet's position is made equal to weird character 2's position and weird character 1's score (S1) is updated. Then the program jumps to line 470 and the explosion sequence takes place.

You should now be familiar with all the elements that make up this exciting game. Take your time and work through the Shootout program to get a feeling for the game's structure, then modify it to your heart's content. Captain Action thinks it's a real winner!

SHOOTOUT AT THE SO-SO CORRAL

10
Player-Missile Graphics Made Simple

Your Atari computer is smarter than you think. It's really two computers tucked cleverly into one sleek and trendy package.

One microprocessor chip does all the things that other computers do. Dull stuff, like adding, subtracting, controlling sound, and basic graphics, and just being a stick-in-the-mud regular "good ole boy." Ho-hum.

The Frantic ANTIC!

The other microprocessor, which we Atari wackos call ANTIC, is Captain Action's favorite. It contains all the pizzazz, all the flash, dash, smash, and futuristic stuff that's lacking in most other computers. The ANTIC microprocessor's primary job is controlling your TV or monitor's display. But here's the exciting news—it also contains what I like to call the Frantic ANTIC component, the part of this superchip that *you* can program to generate Player-Missile graphics!

Four Players and Four Missiles

The Frantic ANTIC chip contains four Players and four Missiles. You can shape each Player exactly as you would a redefined character, assign a color to it, plop it on the screen, and move it around! You can do the same with the four Missiles; each missile's color is the same as its corresponding Player.

What This Means to You

Players and Missiles will:

- Add up to four additional colors to your screen.
- Add up to eight additional movable objects to your screen.
- Move characters faster, without slowing down your program, than BASIC can.
- Add more excitement and pizazz to your game!

Players and Missiles can be displayed in *any* graphics mode, no restrictions!. You design them in ANTIC's memory just as they'll appear on your screen. ANTIC then flips them directly onto the TV screen, regardless of what graphics mode is being displayed.

Things You Can Do with Players and Missiles

You can use Players and Missiles to draw shapes on the screen (they don't always have to move around) and, by taking advantage of the four extra colors, you can make your games come vibrantly alive.

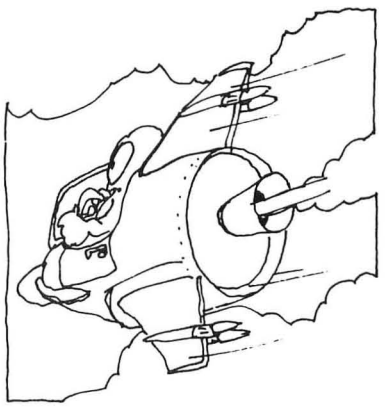
You can use Players and Missiles to add more movable characters to your games, and Player-Missile characters will really zip and zap across your screen!

Because Players and Missiles glide over your screen's display, you can use them as movable cursors. Suppose you want to create a "menu" that lets the user make his or her selection using a joystick to maneuver the cursor over a numbered area. When the cursor is positioned over the selection, the user presses the joystick's trigger and *voila*—the choice is noted by the clever computer and acted upon. Who says you have to communicate with the computer via the keyboard?

Wacko's Player-Missile Miracle Guide

You'll find my Player-Missile Miracle Guide on page 185. This guide is a valuable and necessary tool that you'll refer to often as you learn all about Player-Missile graphics. Right now it might just look like a scrambled set of number-filled charts. Have no fear. Dr. Wacko will show you how to use these charts with complete confidence as you are creating wondrous images on your TV's screen.

To set up and use Player-Missile graphics, you follow a precise set of instructions. I'll explain these instructions step by step, as I take you on a tour of my Player-Missile Miracle Guide. Using Player-Missiles in your arcade games is a snap for bona-fide wackos like me and you!



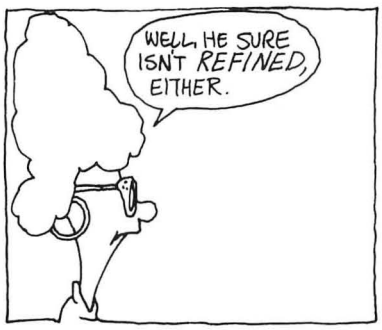
And, after you learn how to place Players and Missiles on your screen, draw with them, and make them zip around, you'll be well on your way toward your Humongous Degree of Computer Wacko Science.

I could spend the next twenty pages talking about all the wonderful things Player-Missiles can do for your games. But the best way to appreciate their power is to watch them in action!

Everything you need to know about Player-Missile graphics is cleverly included in my Know-It-All program below. Enter this short demonstration program, RUN it, watch it, or call in your friends and neighbors to watch it. Then, whenever you're ready, I'll show you how this program works and we'll experiment with some exciting new concepts.

Know-It-All

```
10 GRAPHICS 3+16
20 FOR X=16 TO 24:FOR Y=0 TO 23:COLOR 3:PLOT
   X,Y:NEXT Y:NEXT X
25 .
30 MEMTOP=PEEK(741)+256*PEEK(742)-1
40 PMBASE=INT((MEMTOP-1024)/1024)*1024
50 ADJTOP=PMBASE+384
60 POKE 742,INT(ADJTOP/256):POKE 741,
   ADJTOP-256*PEEK(742)
65 .
```




```
 70 POKE 54279,PMBASE/256
 80 POKE 53277,2
 90 POKE 559,34 + 8
100 P0 = PMBASE + 512
110 FOR A = P0 TO P0 + 128:POKE A,0:NEXT A
120 FOR A = P0 + 60 TO P0 + 67:READ B:POKE
    A,B:NEXT A
130 POKE 53256,3
140 POKE 623,1
150 POKE 704,108
160 POKE 53248,PEEK(20):GOTO 160
170 DATA 60,126,129,153,255,36,66,129
```

Right, Ms. Peeky, Wackenstein is back. But this time, although he's still quite a character, he is not a redefined character.

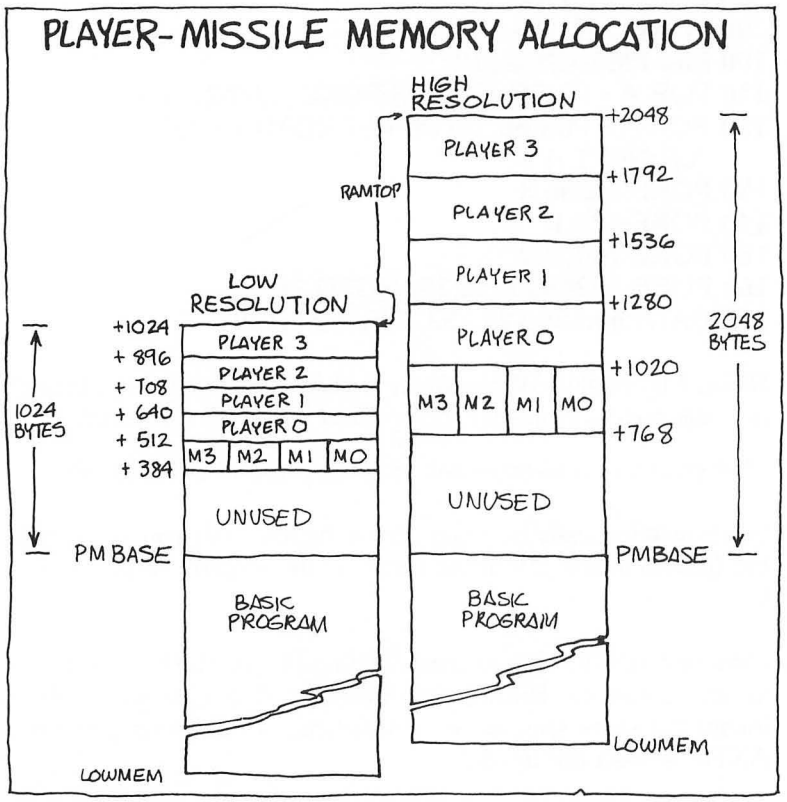
That crazed and bleary-eyed monster is a Player-Missile shape!

See how effortlessly he glides across the screen in front of the blue bar (drawn in line 20) at the center of the screen? Amazing, isn't it?

Using the ANTIC chip to create gliding Player-Missile monsters is an exact science. But it's easy because all you've got to do is follow the *same* specific set of instructions each time you want ANTIC to become frantic!

Here's where my Player-Missile Miracle guide becomes invaluable. Refer to it as we glide through the Know-It-All program.

Player-Missile Memory Allocation



RAMTOP, the top of your Atari's RAM memory, is at the top of these charts, and PMBASE (Short for Player-Missile Base) is at the bottom. So far so good.

I'm going to show you how to squeeze the data that defines your Player-Missiles between RAMTOP and PMBASE. But first, a little chartology.

Low-Resolution Players and Missiles

You can tell ANTIC to display either high- or low-resolution Players and Missiles, depending on how you want them to look and how much memory is available to you.

Using low-resolution Player-Missile shapes in your game programs is a great way to create spiffy screen images (rockets, scenery, and big, big monsters like Wackenstein) without using

lots of memory! However, you *can't* draw really fine, detailed pictures. For those of you who really want to know, the low-resolution option skips every other screen line as it draws each shape on your television.

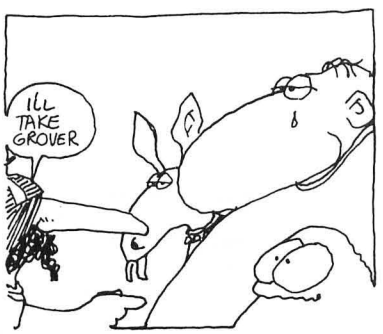
The Low-Resolution Player-Missile Memory Allocation chart shows that you've got to set aside 1024 bytes (1K) of memory for low-resolution Players and Missiles. The data that defines each Player uses 128 bytes of memory. The data for each Missile uses 32 bytes. There is also a 384-byte area marked UNUSED at the bottom of the chart. But I'll show you how to use this UNUSED memory.

High-Resolution Players and Missiles

The high-resolution option packs twice the number of data-filled lines in the same screen area as a low-resolution shape. These additional screen lines let you draw intricately detailed designs and characters. But because high-resolution offers twice the detail, it costs twice as much memory as the low-resolution option. Nothing's cheap these days!

The High-Resolution Player-Missile Memory Allocation Chart shows that you've got to reserve 2048 bytes (2K) of memory for high-resolution Players and Missiles. The data that defines each Hi-Res Player uses 256 bytes of memory. The data for each Missile uses 64 bytes. There is a 768-byte area marked UNUSED at the bottom of this chart too.

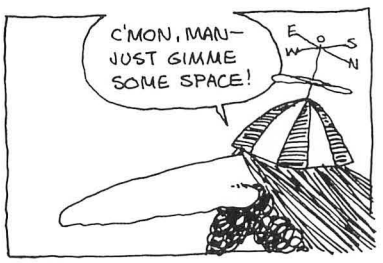
Which Player Do You Want, Anyhow?

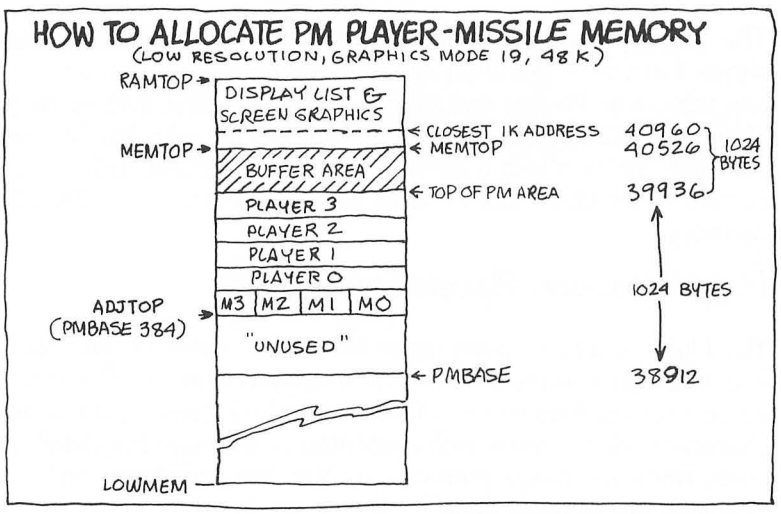


You add the numbers listed at the sides of these charts to PMBASE to tell ANTIC which Player or Missile you want displayed. If you want to display a low-resolution Player 0, just yell "P0 = PMBASE + 512!". ANTIC will get the message! That's what I did in line 100 of the Know-It-All program. Wackenstein is a low-resolution (or low-life) Player 0 character!

Step 1: Make Room for Player-Missiles

Now that you're an expert chartologist, I'll show you how to make room in memory for your nifty players and Missiles.



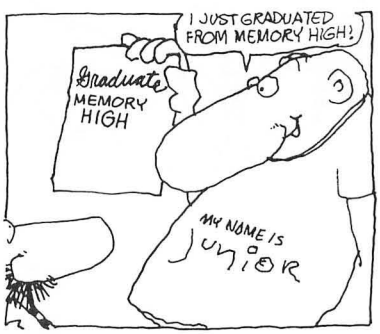


Look carefully at the example. (You are getting sleepy. . .). It shows the actual locations present when memory is set aside for low-resolution Player-Missiles in Graphics Mode 3 + 16 using an 48K Atari 800.

Right below RAMTOP (the top of your computer's Random Access Memory) is the Display List and Graphics Screen memory allocation. Below this is a Buffer Area, followed by the Player-Missile area, then the UNUSED area, and finally your BASIC program.

Line 30: Find out where the Free Memory High Address is.

30 MEMTOP = PEEK(741) + 256 * PEEK(742) – 1



The code in line 30 finds out where the Free Memory High Address is. This memory location, called MEMTOP, is right below the Graphics Screen area you've selected. MEMTOP is location 40526 in the example.

Line 40: Make room for Missiles. When you set up low-resolution Player-Missile graphics, the top of the Player-Missile area will be 1K (1024 bytes) *below* the closest memory location to MEMTOP that is evenly divisible by 1024. Here's how this works out in the example:

MEMTOP is at memory location 40526. If you divide 40526 by 1024 you'll get 39.57617. To find the closest location above MEMTOP that is evenly divisible by 1024 multiply 40 times 1024. It's 40960! Get it? This is important—you are really finding the lowest location within screen memory divisible by 1024, then subtracting 1024 to get out of screen memory.

Next, find the top of the Player-Missile area (location 39936) by subtracting 1024 (1K) from 40960. This takes us out of screen memory.

Finally, subtract 1K (1024 bytes) from location 39936 to arrive at PMBASE.

In the example, PMBASE is at memory location 38912.

Line 40 does all the work:

```
40 PMBASE = INT((MEMTOP - 1024)/1024)*1024
```

I wanted to show you what I'm doing, but don't worry if it's hard to follow, the code in line 40 will do all this work for you!

To make room for high-resolution Player Missiles, replace all the 1024's with 2048 in this line. *High-resolution Player-Missiles must start 2K below the nearest location to MEMTOP that is evenly divisible by 1024.*

Lines 50 and 60: Use that UNUSED area

```
50 ADJTOP = PMBASE + 384
60 POKE 742,INT(ADJTOP/256):POKE 741,
   ADJTOP - 256*PEEK(742)
```

Two very important things are accomplished by making ADJTOP equal to PMBASE + 384 in line 50, and telling your computer that ADJTOP is now the new Free Memory High-Address in line 60:

1. You use the "UNUSED" area

You pick up an additional 384 bytes of usable memory by making ADJTOP equal to PMBASE + 384. You are telling your Atari that it can accept programs right up to the base of the Missiles.

To make room for high-resolution Player-Missiles, change the statement in line 50 to PMBASE + 768 and you'll pick up an additional 768 bytes of usable memory! Such a deal!

2. You protect the Player-Missile area.

Your Player-Missile area is protected from encroachment by invading BASIC programs because you've told your Atari that it can't go past ADJTOP (the new Free Memory high Address you set.)

To sum up. Lines 30 to 60:

- Automatically place the Player-Missile area below any graphics mode you select.
- Protect the Player-Missile area from encroachment by your BASIC programs.
- Increase the usable programming area by using the UNUSED area.

Change the graphics mode in line 10 from 3 + 16 to any mode of your choice to see how spectacular this program is.

Why, you may ask, is this so spectacular? Junior asked me the same thing last week. I answered by showing him how other people have been allocating Player-Missile memory. Every other method known to peoplekind has two weak points:

- When you change graphics modes—from 3 to 7, for example—the display goes haywire because there is no self-adjustment built in.
- If that's not bad enough, those other methods don't protect the Player-Missile area from encroachment. When you write a real long program, the darn thing can sneak into the Player-Missile area and mess it up!

Making room for your Players and Missiles the Wacko Way puts you one step ahead of all those other people. I wrote my Humongous Thesis about it, and just look where it got me!

The best way to fully grasp Player-Missile memory allocation is to draw a blank chart like that in the example, and fill in the memory locations. You can examine memory locations as the computer runs the program by doing the following:

- RUN the Know-It-All program.
- Press BREAK after Wackenstein makes his appearance.
- In the immediate mode, find MEMTOP's location by typing in: PRINT MEMTOP <RETURN>. Then divide MEMTOP by 1024 to see what number you'll multiply by 1024 to find the closest 1K address.
- Find the closest 1K address. Subtract 1024 from this address to find out where the Player-Missile area begins.
- Subtract 1024 from the top of the Player-Missile area to arrive at PMBASE.
- Confirm that you've done everything correctly by typing in: PRINT PMBASE <RETURN>.

Now that you know how to make room for Players and Missiles, let's move on to STEP 2.

Step 2: Tell ANTIC Where PMBASE Is

Simple, since POKEing register 54279 with PMBASE/256 in line 70 tells ANTIC where PMBASE is.

Step 3: Turn On Your Choice of Players Or Missiles, Or Both

The Know-It-All program uses the low-resolution mode and one Player. In Line 80, I've POKEd Register 53277 with 2. This turns on PLAYERS ONLY.

In Line 90, I've POKEd 559 with 34 plus 8. This turns on PLAYERS ONLY. To use ANTIC's high-resolution mode just add 16 to the statement, giving POKE 559,34 + 8 + 16. Refer to the "Turn On" chart for the other POKE vlues that control the number of Players and Missiles.

Step 4: Pick A Player

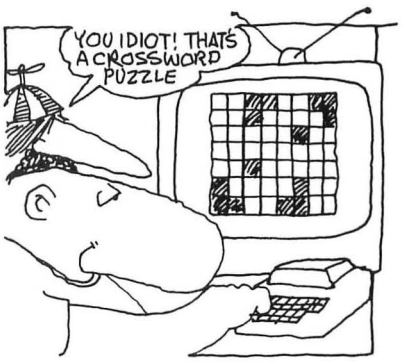
In line 100, Player 0 (P0) begins at PMBASE + 512. (Look back at the Player-Missile Memory Allocation charts.) If you wanted to use Player 1 instead, just change 512 to 640 in this statement.

Step 5: Clear Out the Player Locations

In line 110, I POKE all of Player 0's 128 bytes with zeros. This clears out all of its locations so we can start with a clean slate.

Step 6: Draw the Player's Shape and Position Him Vertically

Line 120 READs Wackenstein's data and POKEs it into Player 0's 60th through 67th bytes.



A low-resolution Player can be up to 128 bytes tall, but Wackenstein is only 8 bytes tall! Unfortunately, if you fill up all 128 bytes, some of the Player's shape will be off your monitor's screen. To see what I mean, replace line 120 with:

120 FOR A = P0 TO P0 + 128:POKE A,255:NEXT

If you want to make a Wackenstein-size Player, rewrite line 120 to read:

120 FOR A = P0 + 60 TO P0 + 67:POKE A,255:NEXT A

To move this 8-byte block vertically, just fill in the 8 bytes at a higher or lower position. To move this solid shape down, for example, change line 120 again to read:

120 FOR A = P0 + 40 TO P0 + 47:POKE A,255:NEXT A

And, to move Wackenstein toward the top of your screen, enter this line 120:

120 FOR A = P0 + 80 TO P0 + 87:READ B:POKE A,B:NEXT A

Move Wackenstein up and down the screen, then experiment with the vertical size and screen location of the solid block!

Step 7: Set the Player's Width

Take a look at the Player's Width chart.

In Line 130, I've POKEd Register 53256 with 3. This sets Wackenstein's width to quadruple size . . . a real fatty! To slim him down a little, just POKE 53256 with 1, and to make him conform to the national average, POKE 53256 with either 0 or 2. Each Player's width is controlled by its own Size Register as shown on the chart.

All four Missiles are controlled by one Size Register—register 53260. The Missile Width chart includes an example of this register's use.

Step 8: Set the Priority Register

Look at the Player vs Playfield Priority chart.

In line 140, I've POKEd register 623 with 1 to make the Player have priority over the playfield. It glides *in front of* the playfield. Change line 140 to read 140 POKE 623,4, and the Player will sneak *behind* the blue line!

Step 9: Set the Player's Color

Look at the Player-Missile Color chart.

In line 150, I've POKEd 704 with 108 to make Wackenstein bright pink. (Blush!) Each Player is assigned its own color register as shown on the chart. POKE Register 704 with any number between 1 and 255 to change Wackenstein's color. How about a Baby-Blue Wackenstein!

Each Missile's color is the same as that of its related Player. Missile 0's color will be the same as Player 0's color; Missile 1's color will be the same as Player 1's color; etc.

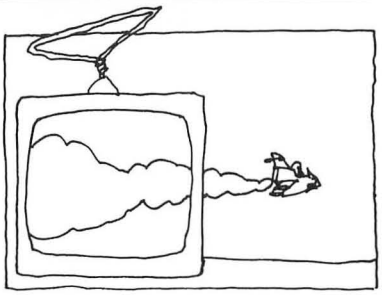
Each Missile's color is the same as that of its related Player. Missile 0's color will be the same as Player 0's color; Missile 1's color will be the same as Player 1's color; etc.

Step 10: Set the Player's Horizontal Position

Look at the Horizontal Position and Collisions chart.

In line 160, I've POKEd register 53248 with PEEK(20). POKEing register 53248 with any number between 0 and 255 will change Player 0's horizontal position.

PEEK(20) counts from 0 to 255, resets, then counts again, to continuously move Player 0.



Although a horizontal position register (like 53248) can accept 256 positions, some of these place the Player or Missile off the left or right edge of the screen (ooops!). If you POKE a horizontal position register with 255, its associated Player or Missile will be off the right edge of the screen, clear out of sight! If you POKE a Horizontal register with 0, its associated Player or Missile will be out of bounds, way off the left side of the screen. To play it safe, only POKE the Horizontal register(s) with values between 60 and 200. That way you'll be sure to see your Players and Missiles.

One trick you can use in your arcade games is to "tell" your Players or Missiles to wait in the wings—off the side of the screen—until you want them to join in the action. When you want a Player to make its grand appearance, just POKE its horizontal register with a number between 60 and 200.

The Ten Player-Missile Commandments

The 10 steps I've shown you must be followed every time you use Player-Missile Graphics. Refer to this list each time you want to make your ANTIC go frantic.

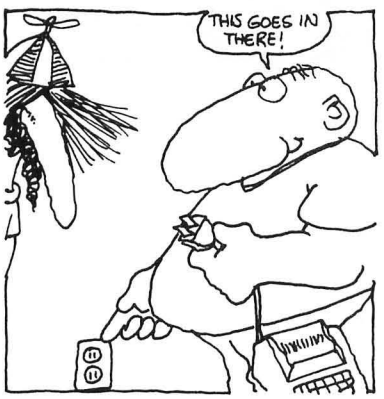
Practice Makes Perfect

The only way to become comfortable with Player-Missile graphics is to practice, practice, and practice. Remember, games are serious business for real wackos!

heavy.
STEP 1: MAKE ROOM FOR PLAYERS AND MISSILES
STEP 2: TELL ANTIC WHERE PMBASE IS
STEP 3: TURN ON YOUR CHOICE OF PLAYERS, MISSILES, OR BOTH
STEP 4: PICK A PLAYER
STEP 5: CLEAR OUT THE PLAYER'S LOCATIONS
STEP 6: DRAW PLAYER'S SHAPE/VERTICAL POSITION
STEP 7: SET THE PLAYER'S WIDTH
STEP 8: SET THE PRIORITY REGISTER
STEP 9: SET THE PLAYER'S COLOR
STEP 10: SET THE PLAYER'S HORIZONTAL POSITION

The Know-It-All program is waiting to be messed with! Here are a few ideas that will help you become truly comfortable with Player-Missile Graphics and Make ANTIC go frantic:

- Make Wackenstein appear as a high-resolution character.
- Use the Monster Maker to create a monster of your own design.
- Display more than one Player.
- Display a low-resolution Missile. Hint: Each Missile is 32 bytes long. POKE PMBASE + 384 to activate Missile 0; PMBASE + 384 + 32 to activate Missile 1, etc.
- Experiment with the width registers. Make your Players and Missiles slim and trim or superhumongous.
- Draw a multicolored skyscraper using a few Players and Missiles.
- Mix COLOR and PLOT drawings with Player-Missile shapes to create multicolored displays.
- Impress yourself, then shock your friends and neighbors with your profound computer wisdom and knowledge.



I've kept my Wackiness in check during this section. After all, you are a graduate student! And Player-Missile graphics does require a certain amount of decorum. But, just so you won't think that I've changed . . . ZAP . . . POW . . . BZZZZRK . . . #$%*!#* + ?#=%!

Super-Snazzy Player-Missile Graphics

Now that you've got a firm understanding of the workings of Player-Missile graphics, its time to get super-snazzy!

Below is an ultracomprehensive Player-Missile program that contains all the elements you need to charge-up your action packed winners:

- Horizontal movement
- Vertical machine-language movement
- Joystick control
- Collision registers
- Brilliant color and vibrant sound
- Many clever programming techniques

The program draws on many of the techniques you've already learned and adds a few new Player-Missile concepts to your repetoire.

Take your time. Type this program in very carefully—it's fragile—one slip and—ooops!

After you've entered it, plug a joystick into port 1, RUN the program, and take control of the flying saucer. Enjoy your flight! When you return to earth, I'll review the program with you step by step. Happy landings!!

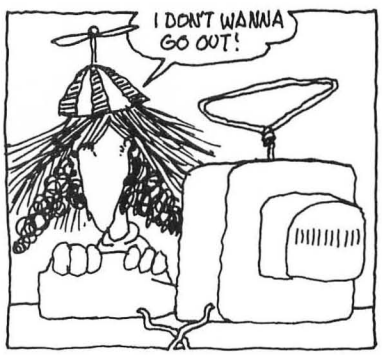
Flying Saucer

```
 10 . Make room for Players & Missiles
 20 GRAPHICS 18
 30 MEMTOP = PEEK(741) + 256*PEEK(742) - 1
 40 PMBASE = INT((MEMTOP - 2048)/2048)*2048
 50 ADJTOP = PMBASE + 768
 60 POKE 742,INT(ADJTOP/256):POKE 741,
    ADJTOP - 256*PEEK(742)
 65 .
 70 SP = ADJTOP - 25:SP1 = ADJTOP - 57:GOSUB 450
 75 .
 80 POKE 54279,PMBASE/256
 90 POKE 53277,2
100 POKE 559,34 + 8 + 16
110 P0 = PMBASE + 1024
115 .
120 FOR A = 0 TO 16
130 B = USR(SP,P0 + A*16,SP1 + 16)
140 NEXT A
145 .
150 POKE 53256,1
160 POKE 623,1
165 .
170 REM Set up Playfield
180 COLOR ASC("—"):PLOT 0,10:DRAWTO 19,10:
    POSITION 2,11:? #6;"PEEK(53252) = ";
190 COLOR 35:PLOT 5,1:PLOT 6,4:PLOT 12,4:PLOT
    5,8:PLOT 12,8
200 COLOR 3:PLOT 8,1:PLOT 5,4:PLOT 13,4:PLOT
    6,7:PLOT 13,7
```

```
210 COLOR 163:PLOT 11,1:PLOT 6,3:PLOT 13,3:PLOT
    6,8:PLOT 14,8
220 COLOR 131:PLOT 14,1:PLOT 7,4:PLOT 14,4:PLOT
    7,8:PLOT 13,8
230 COLOR 49:PLOT 5,0:COLOR 18:PLOT 8,0:COLOR
    180:PLOT 11,0:COLOR 152:PLOT 14,0
235 .
240 X = 116:Y = 185:YB = Y:REM Move Saucer
250 A = STICK(0)
260 DX = (A = 5 OR A = 6 OR A = 7) - (A = 9 OR A = 10
    OR A = 11)
270 DY = (A = 9 OR A = 13 OR A = 5) - (A = 10 OR A = 14
    OR A = 6)
280 X = X + DX*4:Y = Y + DY*4
290 .
300 IF X<48 OR X>192 OR Y<29 OR Y>185 THEN
    X = X - DX*4:Y = Y - DY*4:GOTO 250
310 B = USR(SP,P0 + YB,SP1 + 16)
320 POKE 53248,X
330 B = USR(SP,P0 + Y,SP1)
340 YB = Y
345 .
350 SOUND 0,180 + DY + DX,10,4:SOUND
    1,181 + DY + DX,10,4:POKE 704,PEEK(20)
360 B = PEEK(53252):IF B = 0 THEN POSITION 14,11:?
    #6;"0 ";:GOTO 380
370 POSITION 14,11:? #6;B;" ";:POKE 53278,0
380 GOTO 250
385 .
390 . Machine Language Movement Routine
400 DATA 104, 104, 133, 204, 104, 133, 203, 104, 133,
    207, 104, 133, 206, 160, 0, 177, 206, 145, 203, 200,
    192, 16, 208, 247, 96
405 .
410 . Data for Saucer
420 DATA 24, 24, 24, 24, 24, 60, 24, 255, 255, 24, 60,
    24, 24, 24, 24, 24
430 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
435 .
440 . Read Machine Movement Data
450 RESTORE 400
460 FOR A = 1 TO 25
470 READ B
480 POKE SP + A - 1,B
490 NEXT A
```

```
495 .
500 . Read Saucer Data
510 FOR A = 1 TO 32
520 READ B
530 POKE SP1 + A - 1,B
540 NEXT A
550 RETURN
```

You are about to graduate and receive your Humongous Emeritus Bolonous Diploma. You can leave the hallowed halls of Wacko Institute with a sense of pride and accomplishment—and relief!



Once you have mastered all the elements contained in this program, you can hold your head up high, straighten your back, and march proudly out of your computer room to the world out there.

Armed with the great store of knowledge that you have absorbed during your stay here, you can push forward to conquer even greater challenges, make larger conquests, and, if you're really lucky, get the opportunity to be surrounded by weird cartoon characters, like me.

You'll also be able to design the best arcade games in the universe!! I'll be looking for them at the next User's Group meeting. I feel jealous already!

So, before I hand you your diploma, let's step through this final example of programming brilliance.

Steppin' through Flying Saucer (Don't fall off!)

The flying saucer that you just flew around the screen is a high-resolution Player 0. That's why lines 30 through 60 look so familiar. Also note all those 2048's in line 40, a sure give-away for high-resolution action.

Line 70 was inserted by Snidely Seersucker when I wasn't looking. It's really devious! SP is the location where the machine-language movement data will be stored, and SP1 is the location where the Saucer's data will reside. Because Snidely knows where ADJTOP is, he started the machine-language data table

25 bytes below ADJTOP, and cleverly began the the Saucer's data 32 bytes below the machine-language data. It's amazing what a person can do with a little bit of knowledge!

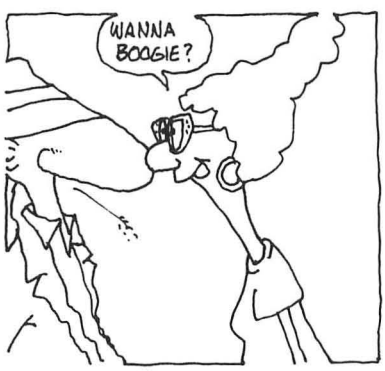
Now that you know where the new top of Free Memory is, you can locate all sorts of stuff below it. Snidely could have protected his 57 bytes of data by redefining ADJTOP to be equal to ADJTOP – SP1. But since this is such a short program I guesss he didn't feel it would be mashed. Don't forget to take this precaution when you see that it's necessary.

The program pauses to do some work at the end of line 70 as it GOSUBs to line 450.

At line 450, the Machine movement Data is read and POKEd into the 25 locations starting with location SP. Once this is done, the saucer data is POKEd into the locations beginning with SP1.

After all of this POKEing around, the program bounces back to line 80.

In line 80, ANTIC is told where PMBASE is hiding. Then in lines 90 and 100, we turn on the high-resolution Players.



Line 100 selects Player 0. So far, so good.

WHAT'S THIS!!! What weird thing is happening in lines 120 to 140? After I stared at it for over two hours, I finally realized what was going on. I had missed dinner!

The Machine-Language Machine snuck in one of its stranger but (now that I think about it) one of its more brilliant USR routines. This ingenious USR routine clears out all of Player 0's 256 bytes by dumping batches of 16 zeros at a time into the Player's memory locations.

This insidious USR routine grabs zeros from SP1 + 16 and dumps them, in batches of 16, into this innocent Player's memory locations—poor guy. Look, and count the data in lines 420 and 430, and you'll get the picture.

Lines 150 and 160 are very straightforward—just your run-of-the-mill Player-Missile stuff. Nothing esoteric here. Line 150 sets Player 0's size register to normal width. In line 160, register 623 is

set so that the Player has priority over the playfield. It's really neat to watch the saucer scoot behind the playfield objects. Just POKE 623 with 3 to check this effect out.

Line 180 draws that dashed line across the bottom of the screen and prints PEEK(53252) = below it. (PEEK(53252) is Player 0's Playfield collision register. You'll see it listed proudly on the Player Missile Horizontal Position and Collision Register chart.

Lines 190 through 220 plot those colored pound signs all over your screen. There *is* a method to the colors used. It may have been a while since you've looked at the graphics chapter, so hunt through that section until you find a chart called Graphics Modes 1 & 2: Color Register Assignments." If you aren't a hunter, just turn to page 34, look at the chart, and follow along.

All those orange #'s are PLOTted with color register 0. The Light Green #'s are color register 1. The Blue #'s are color register 2. The Light Red #'s are color register 3.

Right now is as good a time as any to explain the method behind those pound signs, why the numbers 1, 2, 4 and, 8 appear above them, and what relationship they have to collision register 53252.

When you hotrodded your flying saucer across the screen, I'm sure you flew over one or more pound signs. You couldn't help it, they're all over the place! When you flew over an orange #, the number 1 appeared next to the collision register readout at the bottom of the screen. When you collided with a Blue #, the number 4 appeared. Here's a repeat of the information contained in my Player-Missile Miracle Guide on page 185.

Color Register	Value Bumped Into	Returned
Orange	0	1
Light Green	1	2
Blue	2	4
Light Red	3	8

If you hover the saucer directly above a cluster of all four colors, the number 15 will appear next to the collision register readout—1 + 2 + 4 + 8 = 15!

If you were unlucky enough to touch both a blue and red pound sign at the same time the readout would display the number 12.

When you RUN this program again, keep this handy chart in front of you as you bump into all those pound signs.

Now that you know how the collision registers work, you can use this great Player-Missile feature in your games in place of the LOCATE statement!

Enough diversion. Time to continue our march down the program listing.

Line 230 PLOTs the numbers above the pound signs at the top of the screen.

The code contained in lines 240 through 380 allows you to control the saucer's movement with your joystick and adds a bit of sound and some some flashing colors to the action. Then it reads the collision register and clears itself so it's ready for the next collision.

That was an overview of the "movement" section's major functions. Here's a closer look.

The saucer's starting positions are set in line 240. Then, in lines 250 through 270, it's Bouillabaisse logic to the rescue. If you need a refresher, flip back to Chapter 6, Taking Control Your Joystick, for a second helping of this tasty routine.

The saucer's horizontal movement across the screen is set in line 280. The value of X returned here is POKEd into the horizontal movement register (53248) in line 320 to move the saucer.

If you'd like to see the flying saucer really zip across your screen, replace the two 4's in this line with 10's. Try 20's if you really like to live dangerously! These numbers set the number of pixels the saucer moves horizontally during each movement cycle. But you'll also have to change the 4's in line 300 to be consistent!

Line 300 is your standard "don't go out-of-bounds" and off the screen statement.

The USR routines in lines 310 and 330 are used to move the saucer up and down (vertically). The USR routine in line 330 draws the saucer by dumping its shape from SP1 into P0 + Y, the

Player's Y position. The USR routine in line 310 erases the Player's old position by plopping zeros into the Player's YB position.

Line 340, YB=Y, resets YB equal to Y to end the movement loop.

Line 350 is loaded with weird sound and flashing color. I'll let you ponder these strange SOUND statements. I didn't put them into the program! It looks like the Wacko Cat's handiwork.

POKEing 704 with that clever PEEK(20) turns the saucer into a real flasher.

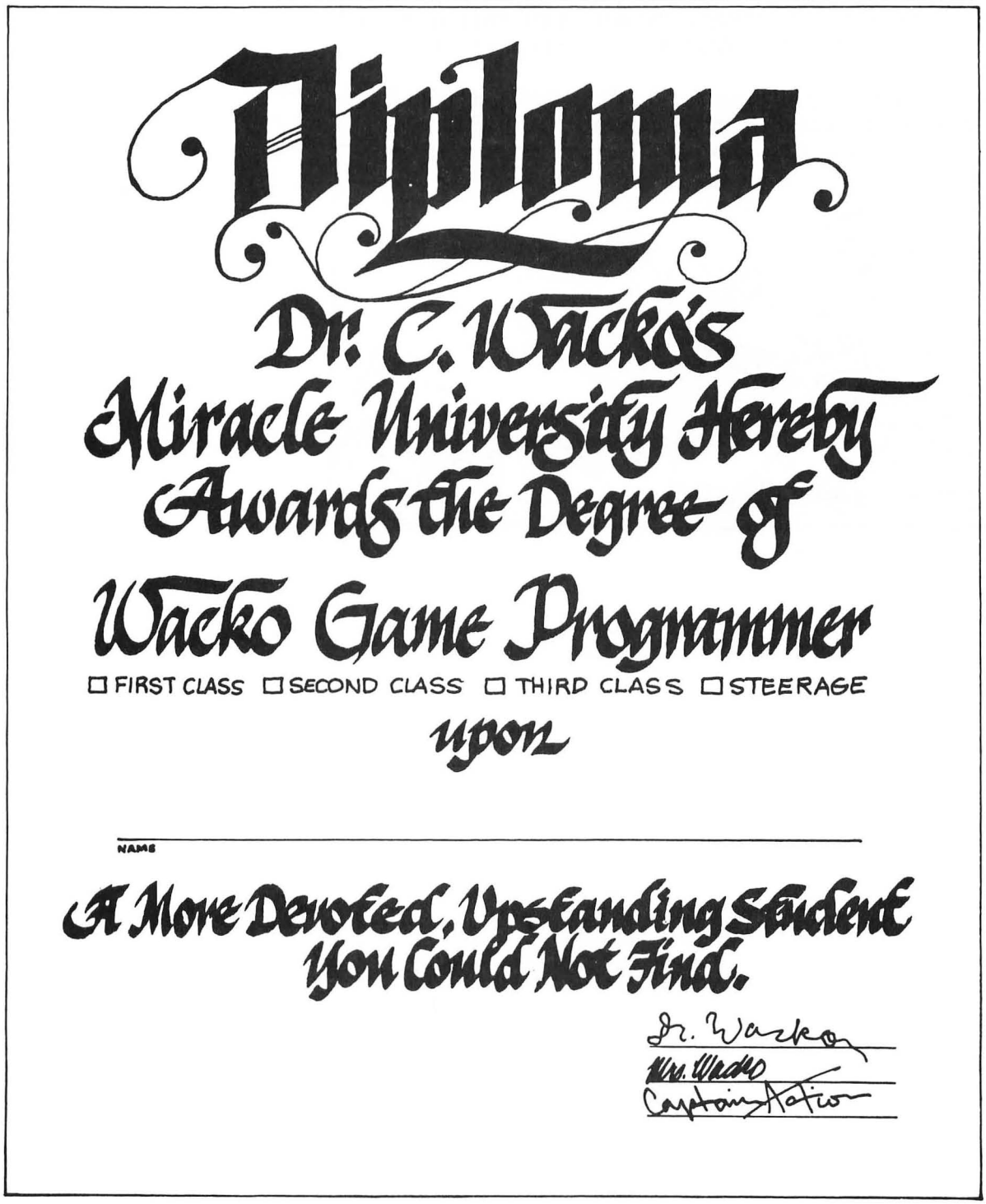
Line 360 takes a look at collision register 53252, and, if the value returned is 0, prints 0 in the readout area and returns, via line 380, to the beginning of the movement routine.

In Line 370, if PEEK 53252 does not equal 0 the collision register's value is printed out in the readout area. Then the collsion register is cleared by POKEing 53278 with 0, and the program loops back to the beginning of the movement routine via the GOTO 250 statement in line 380.

Ba dah, ba dah, ba dah, ba dah. That's all folks!

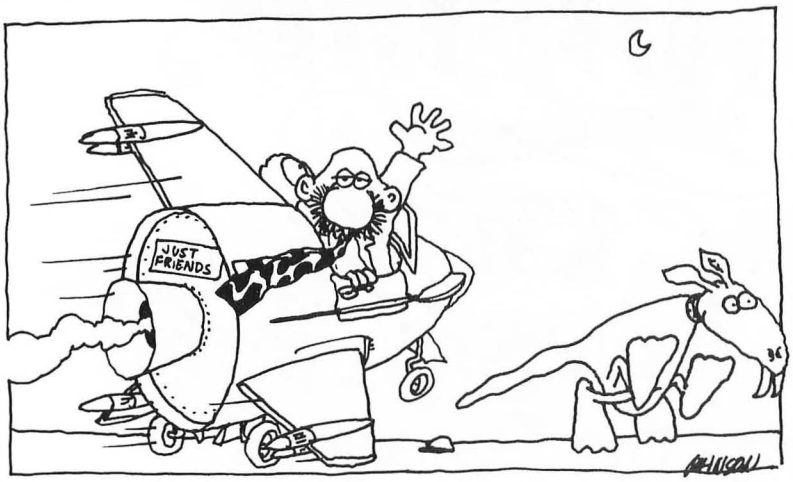
With the exception of some interesting extracurricular activities that follow this chapter, this semester is officially declared complete.

You've earned your diploma. You've done a super job—4.0 average all the way! Turn the page, and award yourself the Dr. C. Wacko Humongous degree of Computer Wacko Science.


Diploma
Dr. C. Wacko's
Miracle University Hereby
Awards the Degree of
Wacko Game Programmer
☐ FIRST CLASS ☐ SECOND CLASS ☐ THIRD CLASS ☐ STEERAGE
upon
NAME
A More Devoted, Upstanding Student
You Could Not Find.
Dr. Wacko
Mr. Wacko
Captain Action

I hope you enjoyed learning all the tricks of the arcade game biz. I certainly enjoyed presenting this exciting material to you.

Until we meet again, this is Dr. C. Wacko signing off from Earth, and (as I ride into the setting sun) saying. . . RIBBIT!!



WACKO'S PLAYER-MISSILE MIRACLE GUIDE

STEP 1: Make Room For Players & Missiles

Low Resolution

1. MEMTOP = PEEK(741) + 256 * PEEK(742)-1
2. PMBASE = INT((MEMTOP-1024)/1024) * 1024
3. ADJTOP = PMBASE + 384
4. POKE 742, INT (ADJTOP/256): POKE 741, ADJTOP-256 * PEEK(742)

*Double the numbers in Lines 2&3 for high resolution players & missiles.

STEP 2: Tell Antic Where PMBASE Is

POKE 54279, PMBASE/256

STEP 3: Turn on Your Choice of Players/Missiles or Both

A. POKE 559, 34 + Value Added

VALUE ADDED	RESULTS (Low Resolution)
4	Turn on Missiles only
8	Turn on Players only
12	Turn on Players & Missiles
ADD 16	High resolution Adder (ADD 16 to the Above to Turn On High Resolution Players & Missiles.

B. POKE 53277, Value

VALUE	RESULTS
1	Turn on Missiles only
2	Turn on Players only
3	Turn on Players & Missiles

STEP 4: Pick a Player

Refer to page 167.

STEP 5: Clear Out the Player's Locations

Example: For A = P1 to P1 + 128: POKE A,0: NEXT A

This line of code clears out Low Resolution Player 1 by POKE-ing 0's into all its locations.

STEP 6: Draw the Player's Shape/Vertical position

See Page 172 for a complete explanation.

STEP 7: Set the Player's Width

POKE size register, width #

Player Width #
0 or 2 = Normal width
1 = Double width
3 = Quadruple width

Player	Size Register
0	53256
1	53257
2	53258
3	53259

Missile	
0-3	53260

Use this chart to set each missile's width.

	Width		
Missile	Normal	Double	Quad.
0	0	1	3
1	0	4	12
2	0	16	48
3	0	64	192

Examples:

1) POKE 53260, 48 to set missile 2's width to quadruple size.
2) POKE 53260, 64 to set missile 3's width to double size.

STEP 8: Set the Priority Register

POKE 623, Value

Value	Results
1	All players have priority over playfield.
2	Players 0 and 1 have priority over playfield and over players 2 and 3.
3	Playfield has priority over all players.
8	Playfield colors 0 and 1 have priority over all players and over playfield registers 2 and 3.

STEP 9: Set the Player's Color

Player & Missile	POKE Color Resister
0	704
1	705
2	706
3	707

Example: POKE 704, 99-Player 0 will be purple

STEP 10: Set the Player's Horizontal position (and Collisions)

Player	WRITE TO Horizontal Pos. Register	READ Collision with Playfield	READ Collision with Player
0	53248	53252	53260
1	53249	53253	53261
2	53250	53254	53262
3	53251	53255	53263
Missile			
0	53252	53248	53256
1	53253	53249	53257
2	53254	53250	53258
3	53255	53251	53259

Collision with Playfield

Color Register Encountered	VALUE Returned
Orange 0	1
Light Green 1	2
Blue 2	4
Light Red 3	8

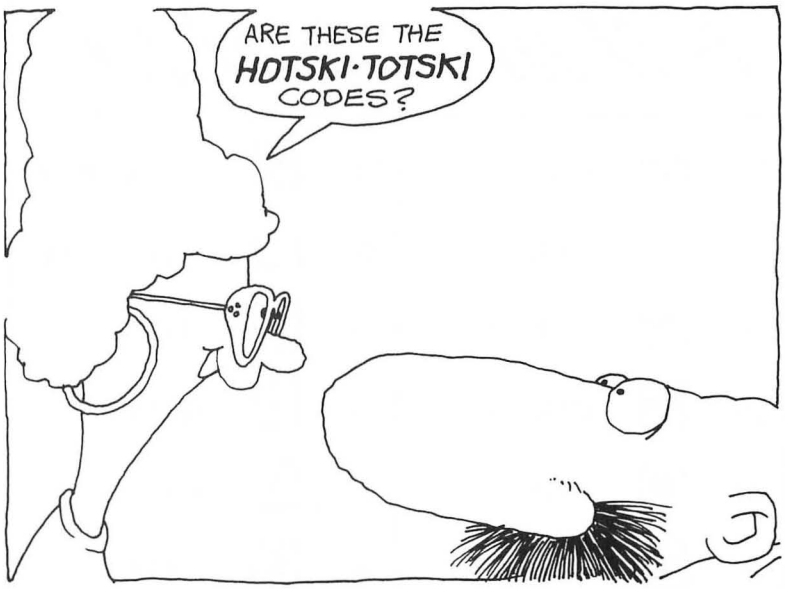
Collision with Player or Missile

Player/Missile Encountered	Value Returned
0	1
1	2
2	4
3	8

To Clear Collision Registers:

POKE 53278,0

Appendix A: ATASCII Codes



Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
0		CTRL-,	8		CTRL-H
1		CTRL-A	9		CTRL-I
2		CTRL-B	10		CTRL-J
3		CTRL-C	11		CTRL-K
4		CTRL-D	12		CTRL-L
5		CTRL-E	13		CTRL-M
6		CTRL-F	14		CTRL-N
7		CTRL-G	15		CTRL-O

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
16		CTRL-P	31		ESC/CTRL-*
17		CTRL-Q	32		SPACE BAR
18		CTRL-R	33		SHIFT-1
19		CTRL-S	34		SHIFT-2
20		CTRL-T	35		SHIFT-3
21		CTRL-U	36		SHIFT-4
22		CTRL-V	37		SHIFT-5
23		CTRL-W	38		SHIFT-6
24		CTRL-X	39		SHIFT-7
25		CTRL-Y	40		SHIFT-9
26		CTRL-Z	41		SHIFT-0
27		ESC/ESC	42		SHIFT-*
28		ESC/CTRL--	43		+
29		ESC/CTRL-=	44		,
30		ESC/CTRL-+	45		-

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
46	.	.	61	=	=
47	/	/	62	>	>
48	0	0	63	?	SHIFT-/
49	1	1	64	@	SHIFT-8
50	2	2	65	A	A
51	3	3	66	B	B
52	4	4	67	C	C
53	5	5	68	D	D
54	6	6	69	E	E
55	7	7	70	F	F
56	8	8	71	G	G
57	9	9	72	H	H
58	:	SHIFT-;	73	I	I
59	;	;	74	J	J
60	<	<	75	K	K

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
76	L	L	91	[	SHIFT-,
77	M	M	92	\	SHIFT-+
78	N	N	93	]	SHIFT-.
79	O	O	94	^	SHIFT-*
80	P	P	95	–	SHIFT--
81	Q	Q	96	♦	CTRL-.
82	R	R	97	a	(LOWR) A
83	S	S	98	b	(LOWR) B
84	T	T	99	c	(LOWR) C
85	U	U	100	d	(LOWR) D
86	V	V	101	e	(LOWR) E
87	W	W	102	f	(LOWR) F
88	X	X	103	g	(LOWR) G
89	Y	Y	104	h	(LOWR) H
90	Z	Z	105	i	(LOWR) I

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
106	j	(LOWR) J	121	y	(LOWR) Y
107	k	(LOWR) K	122	z	(LOWR) Z
108	l	(LOWR) L	123		CTRL-;
109	m	(LOWR) M	124		SHIFT- =
110	n	(LOWR) N	125		ESC/CTRL-< or ESC/SHIFT-<
111	o	(LOWR) O	126		ESC/BACK S
112	p	(LOWR) P	127		ESC/TAB
113	q	(LOWR) Q	128		(^) CTRL-,
114	r	(LOWR) R	129		(^) CTRL-A
115	s	(LOWR) S	130		(^) CTRL-B
116	t	(LOWR) T	131		(^) CTRL-C
117	u	(LOWR) U	132		(^) CTRL-D
118	v	(LOWR) V	133		(^) CTRL-E
119	w	(LOWR) W	134		(^) CTRL-F
120	x	(LOWR) X	135		(^) CTRL-G

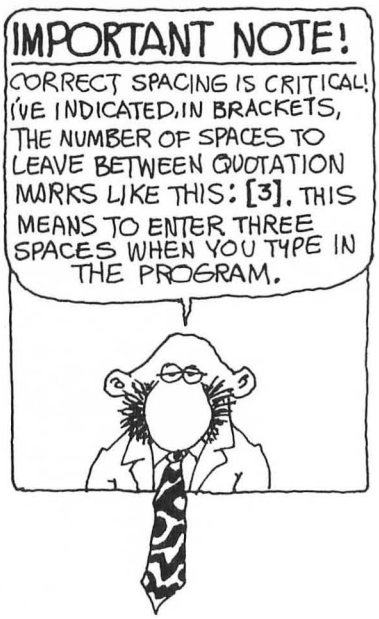
Decimal Code	ATASCII Character	Keys to Create Character
136		(^) CTRL-H
137		(^) CTRL-I
138		(^) CTRL-J
139		(^) CTRL-K
140		(^) CTRL-L
141		(^) CTRL-M
142		(^) CTRL-N
143		(^) CTRL-O
144		(^) CTRL-P
145		(^) CTRL-Q
146		(^) CTRL-R
147		(^) CTRL-S
148		(^) CTRL-T
149		(^) CTRL-U
150		(^) CTRL-V
151		(^) CTRL-W
152		(^) CTRL-X
153		(^) CTRL-Y
154		(^) CTRL-Z
155	EOL	(^) RETURN
156		ESC/SHIFT-BACK S
157		ESC/SHIFT->
158		ESC/CTRL-TAB
159		ESC/SHIFT-TAB
160		(^) SPACE BAR
161		(^) SHIFT-1
162		(^) SHIFT-2
163		(^) SHIFT-3
164		(^) SHIFT-4
165		(^) SHIFT-5

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
166	&	(^) SHIFT-6	181	5	(^) 5
167	'	(^) SHIFT-7	182	6	(^) 6
168	(	(^) SHIFT-9	183	7	(^) 7
169	)	(^) SHIFT-0	184	8	(^) 8
170	*	(^) SHIFT-*	185	9	(^) 9
171	+	(^) +	186	:	(^) SHIFT-;
172	,	(^) ,	187	;	(^) ;
173	-	(^) -	188	<	(^) <
174	.	(^) .	189	=	(^) =
175	/	(^) /	190	>	(^) >
176	0	(^) 0	191	?	(^) SHIFT-/
177	1	(^) 1	192	@	(^) SHIFT-8
178	2	(^) 2	193	A	(^) A
179	3	(^) 3	194	B	(^) B
180	4	(^) 4	195	C	(^) C

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
196		(^) D	211		(^) S
197		(^) E	212		(^) T
198		(^) F	213		(^) U
199		(^) G	214		(^) V
200		(^) H	215		(^) W
201		(^) I	216		(^) X
202		(^) J	217		(^) Y
203		(^) K	218		(^) Z
204		(^) L	219		(^) SHIFT-,
205		(^) M	220		(^) SHIFT-+
206		(^) N	221		(^) SHIFT-.
207		(^) O	222		(^) SHIFT-*
208		(^) P	223		(^) SHIFT--
209		(^) Q	224		(^) CTRL-.
210		(^) R	225		(^) (LOWR) A

Decimal Code	ATASCII Character	Keys to Create Character	Decimal Code	ATASCII Character	Keys to Create Character
226		(INV) (LOWR) B	241		(INV) (LOWR) Q
227		(INV) (LOWR) C	242		(INV) (LOWR) R
228		(INV) (LOWR) D	243		(INV) (LOWR) S
229		(INV) (LOWR) E	244		(INV) (LOWR) T
230		(INV) (LOWR) F	245		(INV) (LOWR) U
231		(INV) (LOWR) G	246		(INV) (LOWR) V
232		(INV) (LOWR) H	247		(INV) (LOWR) W
233		(INV) (LOWR) I	248		(INV) (LOWR) X
234		(INV) (LOWR) J	249		(INV) (LOWR) Y
235		(INV) (LOWR) K	250		(INV) (LOWR) Z
236		(INV) (LOWR) L	251		(INV) CTRL-;
237		(INV) (LOWR) M	252		(INV) SHIFT- =
238		(INV) (LOWR) N	253		(INV) ESC/CTRL-2
239		(INV) (LOWR) O	254		(INV) ESC/CTRL-BACK S
240		(INV) (LOWR) P	255		(INV) ESC/CTRL->

Appendix B: Utility Programs


IMPORTANT NOTE!
CORRECT SPACING IS CRITICAL! I'VE INDICATED, IN BRACKETS, THE NUMBER OF SPACES TO LEAVE BETWEEN QUOTATION MARKS LIKE THIS: [3]. THIS MEANS TO ENTER THREE SPACES WHEN YOU TYPE IN THE PROGRAM.

Color Register POKEs

"You'll be dumbfounded by the wild colors you'll produce."
Dr. C. Wacko

Used with a joystick plugged into port 1, this program will show you all the colors that your Atari can generate.

It is designed to operate in graphics mode 3, and lets you vary color registers 708, 709, 710, and 712. You'll be amazed at the many subtle colors you can create with just simple joystick movement.

Here's how to operate this nifty and colorful utility:

1. Type in and RUN this program.
2. Plug a joystick into port 1.
3. Tilt the joystick in any one of four directions (left, right, up, or down) to *increase* the value that's put into each Color Register.

To *decrease* the value put into each color register, just point the joystick in the desired direction while pressing the red trigger.

Once you've got this program up and RUNning you'll see how easy it is to operate and enjoy.

Color Register POKE Demo

```
10 REM :Wacko's COLOR REGISTER DEMO
20 POKE 764,255:GRAPHICS 3
30 FOR X=3 TO 13:FOR Y=10 TO 17:COLOR 1:PLOT
   X,Y:DRAWTO X+1,Y+1:NEXT Y:NEXT X
40 FOR X=14 TO 23:FOR Y=10 TO 17:COLOR 2
   :PLOT X,Y:DRAWTO X+1,Y+1:NEXT Y:NEXT X
50 FOR X=24 TO 33:FOR Y=10 TO 17:COLOR 3
   :PLOT X,Y:DRAWTO X+1,Y+1:NEXT Y:NEXT X
60 X=10:Y=10:Z=10:Q=10
70 POKE 752,1:? "GRAPHICS 3: Use the Joystick &
   Trigger to see COLOR register POKEs!"
80 FOR P=1 TO 100:NEXT P
   :T=PEEK(644):S=PEEK(632):POKE 752,1
90 IF X>255 OR X<1 THEN X=1:GOTO 80
```

```
100 IF Y>255 OR Y<1 THEN Y = 1:GOTO 80
110 IF Z>255 OR Z<1 THEN Z = 1:GOTO 80
120 IF Q>255 OR Q<1 THEN Q = 1:GOTO 80
130 IF S = 13 AND T = 0 THEN Q = Q - 1:POKE 712,Q:
    ? CHR$(125):? :? CHR$(127);"BACKGROUND
    COLOR: POKE 712,";Q:GOTO 80
140 IF S = 11 AND T = 0 THEN X = X - 1:POKE 708,X:
    ? CHR$(125);" 708,";X:GOTO 80
150 IF S = 14 AND T = 0 THEN Y = Y - 1:POKE 709,Y:
    ? CHR$(125);CHR$(127);CHR$(127);"[2]709,"
    ;Y:GOTO 80
160 IF S = 7 AND T = 0 THEN Z = Z - 1:POKE 710,Z:
    ? CHR$(125);CHR$(127);CHR$(127);CHR$(127);
    "[3]710,";Z:GOTO 80
170 IF S = 11 THEN X = X + 1:POKE 708,X:? CHR$(125);
    " 708,";X:GOTO 80
180 IF S = 14 THEN Y = Y + 1:POKE 709,Y:
    ? CHR$(125);CHR$(127);CHR$(127);"[2]709,"
    ;Y:GOTO 80
190 IF S = 7 THEN Z = Z + 1:POKE 710,Z:
    ? CHR$(125);CHR$(127);CHR$(127);CHR$(127);
    "[3]710,";Z:GOTO 80
200 IF S = 13 THEN Q = Q + 1:POKE 712,Q:
    ? CHR$(125):? :? CHR$(127);"BACKGROUND
    COLOR: POKE 712,";Q:GOTO 80
210 GOTO 80
```

ATASCII Code Program

Here it is. The program that shows you what's going on inside the your computer. ATASCII Codes is easy to use, and will help you understand how your Atari computer generates its cast of characters.

Just type in the program, RUN it, and follow the simple instructions.

Simple Instructions

Refer to the ATASCII chart in Appendix A. Enter the decimal number assigned to the character that you'd like to examine, and press RETURN. For example, if you want to check out the letter A, enter 65 <RETURN>.

You'll be presented with the character's appearance, its offset number, and the bytes that define it.

Press START to look at another character.

Important programming note: Enter all underlined words and characters between quotation marks as inverse characters. To do this, press the Atari symbol key before typing the character. An example of inverse characters is found in line 50.

ATASCII Codes

```
 10 REM THIS PROGRAM WILL CALCULATE OFFSET
    FOR ANY ATASCII DECIMAL CODE
 20 POKE 764,255:POKE 77,0
 30 GRAPHICS 0:POKE 752,1:POKE 710,128:POSITION
    7,3:PRINT "ENTER ATASCII DECIMAL CODE";
 40 TRAP 30:INPUT C
 50 ? :? CHR$(127);CHR$(127);"CHARACTER:";CHR$(C)
 60 IF C>255 THEN GOTO 30
 70 IF C<32 THEN D = (C + 64) * 8
 80 IF C>127 AND C<160 THEN D = (C - 64) * 8
    :GOTO 160
 90 IF C>31 AND C<96 THEN D = (C - 32) * 8
100 IF C>159 AND C<224 THEN D = (C - 160) * 8
    :GOTO 160
110 IF C>95 AND C<128 THEN D = C * 8
120 IF C>223 AND C<256 THEN D = (C - 128) * 8
    :GOTO 160
130 ? :? CHR$(127);CHR$(127);"[2]OFFSET = ";D
140 ?
150 ? :? CHR$(127);CHR$(127);"ROW[5]DATA":? :FOR
    A = 0 TO 7:? CHR$(127);CHR$(127);
    " ";A;"--------";PEEK(D + A + 57344)
155 NEXT A:GOTO 170
160 ? :? "[13]ROW[7]DATA":FOR A = 0 TO 7:?
    "[1]";A;"--------";255 - PEEK(D + A + 57344):NEXT A
170 POSITION 7,21:? "[2]PRESS START TO CONTINUE"
180 IF PEEK(53279)<>6 THEN GOTO 170
190 GOTO 30
```

Monster Maker

Here is the listing for the greatest character-designing program in the world. Instructions for Monster Maker can be found on page

63. Important programming note: Enter all underlined words and characters between quotation marks as inverse characters. To do this, press the Atari symbol key before typing the character. An example of underlined characters is found in line 20: PLEASE WAIT.

Monster Maker

```
  5 REM MONSTER MAKER - Written by David L.
    Heller & Robert Kurcina, Copyright 1983, Addison-
    Wesley Publishing
 10 POKE 764,255
 20 GRAPHICS 0:POKE 752,1:POKE 710,128:
    POSITION 13,11:? "PLEASE WAIT ";:POKE 712,134
 30 DIM N$(15),NA$(15),M$(25),T$(18),CH$(12),
    L$(10),D$(5)
 35 CH$ = CHR$(156):L$ = CHR$(30):D$ = CHR$(29)
 40 M$ = "Insert Machine Language Movemet Routine;
    Chapter 4, Page 75. Note: 22nd Byte is 255 (CTRL,
    >)."
 50 ST = (PEEK(742) - 4)*256:FOR A = 0 TO
    3:Z = USR(ADR(M$),ST + A*256,57344 + A*256)
    :NEXT A
 60  MAXLOCATIONS = 64:X = 0:Y = 0:C = 160
    :CB = 32:L = 0:DIM VALU(MAXLOCATIONS,8),
    BITADD(8),VLOC(8),A$(1)
 70 FOR A = 1 TO MAXLOCATIONS:FOR B = 1 TO 8
    :VALU(A,B) = 0:NEXT B:NEXT A
 80 GOSUB 1420
 90 FOR A = 0 TO 7:POKE ST + 80 + A,85:POKE
    ST + 8 + A,170:POKE ST + 40 + A,0:NEXT A
100 CLOSE #2:OPEN #2,4,0,"K:":GRAPHICS 0
110 ? CHR$(125);:POKE 752,1:POKE 756,ST/256:POKE
    709,10:POKE 712,128:POKE 710,0
120 POSITION 15,0:? "Dr. C WACKO'S":POSITION
    14,1:? "MONSTER MAKER"
130  POKE 16,64:POKE 53774,64
140 ? CHR$(127);"!!!!!!!!!![15]O = ORIG"
150 ? CHR$(127);"!      ![2] = [1]0[8]H/V = FLIPS"
160 ? CHR$(127);"!      ![2] = [1]0[4]U/D/L/R = ROLLS"
170 ? CHR$(127);"!      ![2] = [1]0[10]P = PRINT"
180 ? CHR$(127);"!      ![2] = [1]0[5]OPTION = RVS"
190 ? CHR$(127);"!      ![2] = [1]0[5]SELECT = CLR"
```

```
200 ? CHR$(127);"!      ![2] = [1]0[6]START = EXIT"
210 ? CHR$(127);"!      !2][ = [1]0[4]"
220 ? CHR$(127);"!      ![2] = [1]0[4]"
230 ? CHR$(127);"!!!!!!!!!!"
240 POSITION 3,4:? "%";L$;L$;D$;D$;"%%%";L$;
    L$;L$;D$;D$;"%%%";L$;L$;L$;D$;"%%%";:
    POSITION 0,10
250 POSITION 5,14:? "1 --> EDIT LOCATION"
260 POSITION 5,15:? "2 --> COPY LOCATION"
270 POSITION 5,16:? "3 --> SAVE FONTLIST"
280 POSITION 5,17:? "4 --> LOAD OLD FILE"
290 POSITION 5,18:? "5 --> LIST FNTFILES"
300 TRAP 110:POSITION 13,21:? "[3]OPTION[5]";
    :POSITION 22,21:INPUT A:IF A<1 OR A>5THEN
    300
310 IF A = 1 THEN POSITION 13,21:?
    "FOR LOCATION";:POSITION 25,21:INPUT L
320 TRAP 110:TRAP 110:IF A = 2 THEN POSITION
    8,21:? "FROM ————, TO ———— : ";
    :POSITION 27,21:INPUT L,L5
330 IF A = 1 THEN POSITION 13,21:? "":POSITION
    25,14:? " #";L;
340 ON A GOTO 740,660,440,360,540
350 STOP
360 POSITION 15,21:? CHR$(253);"[1]LOAD IT";:INPUT
    A$:IF A$<>"Y" THEN 110
370 POSITION 15,22:? "C: OR D:NAME";
380 INPUT NA$:IF NA$ = "C:" THEN GOTO 400
385 IF NA$(LEN(NA$) - 3,LEN(NA$))<>".FNT" THEN
    NA$(LEN(NA$) + 1,LEN(NA$) + 4) = ".FNT"
390 IF NA$(1,2)<>"D:" OR LEN(NA$)>14 OR
    LEN(NA$)<7 THEN N$ = NA$:GOTO 1380
400 TRAP 1390:POSITION 24,9:? "LOADING[1]":
    POSITION 24,10:? NA$:CLOSE #1:OPEN
    #1,4,0,NA$:GET #1,MAXLOCATIONS
410 FOR A = 1 TO MAXLOCATIONS:FOR B = 1 TO
    8:GET #1,Z:VALU(A,B) = Z:POKE 712,Z:NEXT
    B:NEXT A:CLOSE #1
420 POKE 712,70:IF MAXLOCATIONS<64 THEN FOR
    A = MAXLOCATIONS + 1 TO 64:FOR B = 1 TO
    8:VALU(A,B) = 0:NEXT B:NEXT A
430 MAXLOCATIONS = 64:? CHR$(253):GOTO 110
440 POSITION 15,21:? CHR$(253);"[1]SAVE IT";:INPUT
    A$:IF A$<>"Y" THEN 110
```

```
450 POSITION 16,21:? "CHARACTERS[4]";:TRAP
    1400:POSITION 26,21:INPUT A:IF A<1 OR A>64
    THEN 110
460 MAXLOCATIONS = A
470 POSITION 13,22:? "C: OR D:NAME";
480 INPUT N$
485 IF N$ = "C:" THEN GOTO 520
490 TRAP 1380:IF N$(LEN(N$) - 3,LEN(N$) - 3)<>"."
    THEN N$(LEN(N$) + 1,LEN(N$) + 4) = ".FNT":
    GOTO 510
500 IF N$(LEN(N$) - 3,LEN(N$))<>".FNT" THEN 1380
510 IF N$(1,2)<>"D:" OR LEN(N$)>14 OR LEN(N$)<3
    THEN GOTO 1380
520 POSITION 24,9:? "SAVING[1]":POSITION 24,10:
    ? N$:CLOSE #1:OPEN #1,8,0,N$:PUT
    #1,MAXLOCATIONS
530 FOR A = 1 TO MAXLOCATIONS:FOR B = 1 TO
    8:PUT #1,VALU(A,B):NEXT B:NEXT A:CLOSE #1:
    ? CHR$(253):GOTO 110
540 CLOSE #1:OPEN #1,6,0,"D:*.FNT":POSITION
    0,12:GOSUB 1500:A = 0
550 TRAP 620:INPUT #1;NA$
560 IF LEN(NA$)>4 AND NA$(4,4) = "[1]" THEN CLOSE
    #1:GOTO 630
570 POSITION 10*(A - INT(A/4)*4),
    12 + INT(A/4):PRINT NA$(1,10);
580 A = A + 1:IF A>31 THEN 600
590 GOTO 550
600 POSITION 10,22:?"PRESS START TO CONT";:IF
    PEEK(53279)<>6 THEN 600
610 A = 0:POSITION 0,12:GOSUB 1500:GOTO 550
620 CLOSE #1
630 POSITION 15,22:? "PRESS START";
640 IF PEEK(53279)<>6 THEN 640
650 GOTO 110
660 IF L5<1 OR L5>64 OR L>64 OR L< - 127
    THEN 110
670 POSITION 0,21:? CHR$(156);CHR$(156);
    CHR$(156);:POSITION 25,15:? L;"[1]to[1]";
    L5:POSITION 25,14:? "[1]#";L5
675 POKE 16,64:POKE 53744,64:POKE 710,97:POKE
    712,101:POKE 752,1
680 IF L>0 THEN FOR Z = 0 TO
    7:VLOC(Z + 1) = VALU(L,Z + 1):NEXT Z:L = L5:
    GOTO 790
```

```
 690 L = ABS(L)
 700 IF L>95 THEN 730
 710 IF L>31 THEN L = L - 32:GOTO 730
 720 L = L + 64
 730 A = L*8 + 57344:FOR Z = 0 TO
     7:VLOC(Z + 1) = PEEK(A + Z):NEXT Z:L = L5:
     GOTO 790
 740 IF L<1 OR L>64 THEN 110
 750 FOR A = 0 TO 7:VLOC(A + 1) = VALU(L,A + 1):NEXT
     A:GOTO 790
 760 IF L<1 OR L>MAXLOCATIONS THEN 110
 770 FOR Z = 0 TO 7:A = 255 - VLOC(Z + 1):
     VLOC(Z + 1) = A
 780 POSITION 21,Z + 3:? A;"[2]";:GOTO 800
 790 FOR Z = 0 TO 7:A = VLOC(Z + 1):POSITION
     21,Z + 3:? A;"[2]";
800  POKE ST + Z + 40,A:B = 0
 810 IF A<1 THEN 840
 820 IF A - BITADD(B + 1)<0 THEN B = B + 1:GOTO 820
 830 COLOR 42:PLOT 8 + B,Z + 3:A = A - BITADD(B + 1)
     :B = B + 1:GOTO 810
 840 NEXT Z
 850 POKE 77,0:LOCATE X + 8,Y + 3,C:IF C = 42 THEN
     COLOR 147:PLOT X + 8,Y + 3
 860 IF C = 32 THEN COLOR 147:PLOT X + 8,Y + 3
 870 FOR A = 1 TO 10:NEXT A
 880 COLOR C:PLOT X + 8,Y + 3
 890 A = STICK(0):B = STRIG(0)
 900 IF B = 0 AND C = 42 THEN COLOR 32:PLOT
     X + 8,Y + 3:VLOC(Y + 1) = VLOC(Y + 1) - BITADD
     (X + 1):POKE ST + Y + 40,VLOC(Y + 1)
 910 IF B = 0 AND C = 32 THEN COLOR 42:PLOT
     X + 8,Y + 3:VLOC(Y + 1) = VLOC(Y + 1) + BITADD
     (X + 1):POKE ST + Y + 40,VLOC(Y + 1)
 920 IF B = 0 THEN SOUND 0,100,10,6:POSITION
     21,Y + 3:? VLOC(Y + 1);"[2]";:SOUND 0,0,0,0
 930 POSITION 27,9:PRINT "CTRL + M = MENU"
 940 IF PEEK(764) = 165 THEN CLR :RUN "D:MENU"
 950 IF A = 7 THEN X = X + 1
 960 IF A = 11 THEN X = X - 1
 970 IF A = 13 THEN Y = Y + 1
 980 IF A = 14 THEN Y = Y - 1
 990 IF A = 10 THEN X = X - 1:Y = Y - 1
1000 IF A = 9 THEN X = X - 1:Y = Y + 1
1010 IF A = 6 THEN X = X + 1:Y = Y - 1
```

```
1020 IF A = 5 THEN X = X + 1:Y = Y + 1
1030 IF X<0 THEN X = 7
1040 IF X>7 THEN X = 0
1050 IF Y<0 THEN Y = 7
1060 IF Y>7 THEN Y = 0
1070 POSITION 27,4
1080 IF PEEK(53279) = 3 THEN GOSUB 1220:GOTO 760
1090 IF PEEK(53279) = 5 THEN GOSUB 1220
     :GOSUB 1210
1100 IF PEEK(53279) = 6 THEN FOR A = 0 TO
     7:VALU(L,A + 1) = VLOC(A + 1):NEXT A:SOUND
     0,0,0,0:GOTO 110
1110 IF PEEK(764) = 255 THEN 1200
1120 GET #2,A:IF A = ASC("H") THEN POKE
     764,255:GOTO 1230
1130 IF A = ASC("V") THEN POKE 764,255:GOTO 1260
1140 IF A = ASC("P") THEN POKE 764,255:GOSUB 1270
1150 IF A = ASC("L") THEN POKE 764,255:GOTO 1290
1160 IF A = ASC("R") THEN POKE 764,255:GOTO 1310
1170 IF A = ASC("U") THEN POKE 764,255:GOTO 1330
1180 IF A = ASC("D") THEN POKE 764,255:GOTO 1350
1190 IF A = ASC("O") THEN POKE 764,255:GOTO 1370
1200 GOTO 850
1210 FOR A = 0 TO 7:VLOC(A + 1) = 0:POSITION
     21,A + 3:? "0[2]";:NEXT A:RETURN
1220 FOR A = 0 TO 7:POSITION 8,3 + A:? "[8]";:POKE
     ST + A + 40,0:NEXT A:RETURN
1230 FOR A = 0 TO 7:VLOC(A + 1) = 0:FOR B = 0 TO
     7:LOCATE B + 8,A + 3,C:IF C = 42 THEN
     VLOC(A + 1) = VLOC(A + 1) + BITADD(8 - B)
1240 NEXT B:NEXT A:GOSUB 1220
1250 GOTO 790
1260 FOR A = 0 TO 7:VALU(0,A + 1) = VLOC(8 - A):NEXT
     A:FOR A = 0 TO 7:VLOC(A + 1) = VALU(0,A + 1)
     :NEXT A:GOSUB 1220:GOTO 790
1270 FOR A = 2 TO 11:FOR B = 7 TO 24:LOCATE
     B,A,C:T$(B - 6) = CHR$(C):NEXT B:LPRINT
     T$:NEXT A
1280 LPRINT "LOCATION[1]";L:LPRINT :RETURN
1290 FOR A = 0 TO 7:VLOC(A + 1) = VLOC(A + 1)*2
     :IF VLOC(A + 1)>255 THEN
     VLOC(A + 1) = VLOC(A + 1) - 255
1300 NEXT A:GOSUB 1220:GOTO 790
1310 FOR A = 0 TO 7:B = VLOC(A + 1)/2:IF B<>INT(B)
     THEN B = INT(B) + 128
```

```
1320 VLOC(A + 1) = B:NEXT A:GOSUB 1220:GOTO 790
1330 B = VLOC(1):FOR A = 0 TO
     6:VLOC(A + 1) = VLOC(A + 2):NEXT A:VLOC(8) = B
1340 GOSUB 1220:GOTO 790
1350 B = VLOC(8):FOR A = 6 TO 0 STEP
      - 1:VLOC(A + 2) = VLOC(A + 1):NEXT A:VLOC(1) = B
1360 GOSUB 1220:GOTO 790
1370 FOR A = 0 TO 7:VLOC(A + 1) = VALU(L,A + 1):NEXT
     A:GOSUB 1220:GOTO 790
1380 POSITION 24,9:PRINT "BAD FILE NAME:"
     :POSITION 24,10:? N$:FOR P = 0 TO 500:NEXT
     P:GOTO 110
1390 CLOSE #1:MAXLOCATIONS = 64:POSITION 24,9:?
     "I CAN'T FIND:":POSITION 24,10:? NA$
1400 FOR A = 1 TO 500:IF PEEK(53279) = 6 THEN POP
     :GOTO 110
1410 NEXT A:GOTO 110
1420 A = 0:B = 0:C = 256:FOR A = 1 TO 8:B = C:FOR D = 1
     TO A:B = B/2:NEXT D:BITADD(A) = B:VLOC(A) = 0
     :NEXT A:RETURN
1500 FOR A = 1 TO 12:PRINT CH$;:NEXT A: RETURN
```

Animation Tester

This program will allow you to see your demented creations cavort in animated action on your screen. Complete operating instructions for Animation Tester can be found on page 71. Important programming note: Enter all underlined words and characters between quotation marks as inverse characters. To do this, press the Atari symbol key before typing the characters. An example of underlined characters is found in line 110.

Animation Tester

```
 10 POKE 764,255
 20 REM ANIMATION TESTER - Written by David L.
    Heller & Robert Kurcina, Copyright 1983, Addison-
    Wesley Publishing
 30 CLR :DIM A$(25),B$(25),C$(14),NA$(25),U$(35)
 35 U$ = CHR$(156)
 40 A$ = "Insert Machine Language Movement Routine;
    Chapter 4. Note: 22nd Byte is 255 (CTRL,>)"
 50 SP = ADR(A$)
 60 GOSUB 470:TRAP 640
 70 ST = (PEEK(742) - 4) * 256
 80 FOR A = 0 TO 3
 90 D = USR(SP,ST + A * 256,57344 + A * 256)
100 NEXT A
110 IF C$ = "C:" THEN CLOSE #1:GRAPHICS 17
    :POSITION 3,3:? #6;"cassette users":POSITION
    5,10:? #6;"PRESS PLAY"
120 IF C$ = "C:" THEN POSITION 8,12:
    ? #6;"THEN":POSITION 5,14:? #6;"HIT return"
    :GOTO 140
130 CLOSE #1
140 OPEN #1,4,0,C$
150 GET #1,A
160 FOR B = 1 TO A
170 FOR C = 1 TO 8
180 GET #1,D
190 POKE ST + B * 8 + C - 1,D
200 NEXT C
210 NEXT B
220 CLOSE #1
230 GRAPHICS G:POKE 752,1:POKE 710,0:POSITION
    3,0:? #6;C$;
```

```
240 POKE 756,ST/256
250 ? :? "[5]PRESS START TO TRY ANOTHER"
260 ? "[5]PRESS SELECT FOR MENU"
270 FOR B = 1 TO A
280 COLOR B + 32
290 PLOT 9,5
300 SOUND 0,0,10,10:FOR C = 1 TO S
310 NEXT C:IF PEEK(53279) = 6 THEN RUN
320 IF PEEK(53279) = 5 THEN CLR :RUN "D:MENU"
330 SOUND 0,0,0,0:NEXT B
340 SOUND 0,0,0,0:GOTO 270
350 CLOSE #1:OPEN #1,6,0,"D:*.FNT":POSITION 0,12:
    ? U$;U$;U$;U$;U$;U$;U$;U$;U$;U$;U$;U$;:A = 0
360 TRAP 430:INPUT #1;NA$
370 IF LEN(NA$)>4 AND NA$(4,4) =
    "[1]" THEN CLOSE #1:GOTO 440
380 POSITION 10*(A - INT(A/4)*4),12 + INT(A/4):
    PRINT NA$(1,10);
390 A = A + 1:IF A>31 THEN 410
400 GOTO 360
410 POSITION 10,22:? "PRESS START TO CONT";:
    IF PEEK(53279)<>6 THEN 410
420 A = 0:POSITION 0,12:
    ? U$;U$;U$;U$;U$;U$;U$;U$;U$;U$;U$;U$;:
    GOTO 360
430 CLOSE #1
440 POSITION 15,22:? "PRESS START";
450 IF PEEK(53279)<>6 THEN 450
460 POSITION 15,22:? "[11]":POKE 764,255:GOTO 520
470 GRAPHICS 0:POKE 710,128:POKE 712,148:
    POSITION 14,6:? "DR. C. WACKO'S":
    POSITION 13,7:? "ANIMATION TESTER"
480 POKE 752,1:POSITION 5,10:PRINT
    "WANT LIST OF FONTS? YES OR NO:"
490 IF PEEK(764) = 43 THEN POKE 764,255:GOTO 350
500 IF PEEK(764) = 35 THEN POKE 764,255:GOTO 520
510 IF PEEK(764)<>43 OR PEEK(764)<>35 THEN
    GOTO 490
520 POSITION 0,12:? U$;U$;U$;U$:POSITION 8,12:
    ?"C: OR D:FONT NAME";:INPUT C$:IF C$ = "C:"
    THEN 550
530 IF LEN(C$)<2 THEN 520
540 TRAP 520:IF C$(LEN(C$) - 3,LEN(C$))<>".FNT"
    THEN C$(LEN(C$) + 1,LEN(C$) + 5) = ".FNT"
```

```
550 POSITION 0,14:? U$;U$:POSITION 6,14:
    ? "GRAPHICS MODE: 0, 1, OR 2 ";
560 TRAP 540:INPUT G
570 IF G = 0 THEN V = 0
580 IF G = 1 THEN V = 1
590 IF G = 2 THEN V = 2
600 IF G<0 OR G>2 THEN GOTO 540
610 POSITION 0,16:? U$;U$;U$:POSITION 16,16:
    ?"Fast Slow":? "[6]SPEED: ( 5 TO 500 ): ";
620 TRAP 600:INPUT S
630 POKE 559,0:RETURN
640 POKE 559,34:PRINT CHR$(125):POKE 752,1:
    POKE 710,53:POSITION 8,10:? "I CAN'T
    FIND[1]";C$;:
    FOR A = 1 TO 500
650 NEXT A:GOTO 60
```

Sound Machine

After you RUN the Sound Machine a list of numbered options will appear on your screen. Just type in a number and listen to the amazing and weird sounds.

The Sound Machine program lets you place any of its sound effects into your program. For example, lines 9000 through 9090 (STAR RAIDERS) can be used as a subroutine in a space game of your own design.

Creating Your Own Sounds

Enter number 13 and press RETURN and the Sound Machine flips to Sound Dabbler MK.I—its creative mode. In this mode you'll be able to astound your parrakeets with strange birdlike chirps, or frighten your neighbors with wild screeches. Anything is possible!

If you want to mix two voices you'll need two joysticks. One plugged into port 1 and the other in port 2.

To change PITCH: Move joystick up or down.

To change DISTORTION: Move joystick left or right.

To vary the VOLUME: Press the red trigger button.

Press OPTION to reset voice 0

Press SELECT to reset voice 1

Press START to exit the Sound Dabbler.

Important programming note: Enter all underlined words and characters between quotation marks as inverse characters. To do this, press the Atari symbol key before typing the characters. An example of underlined characters is found in line 110: THE.

Sound Machine

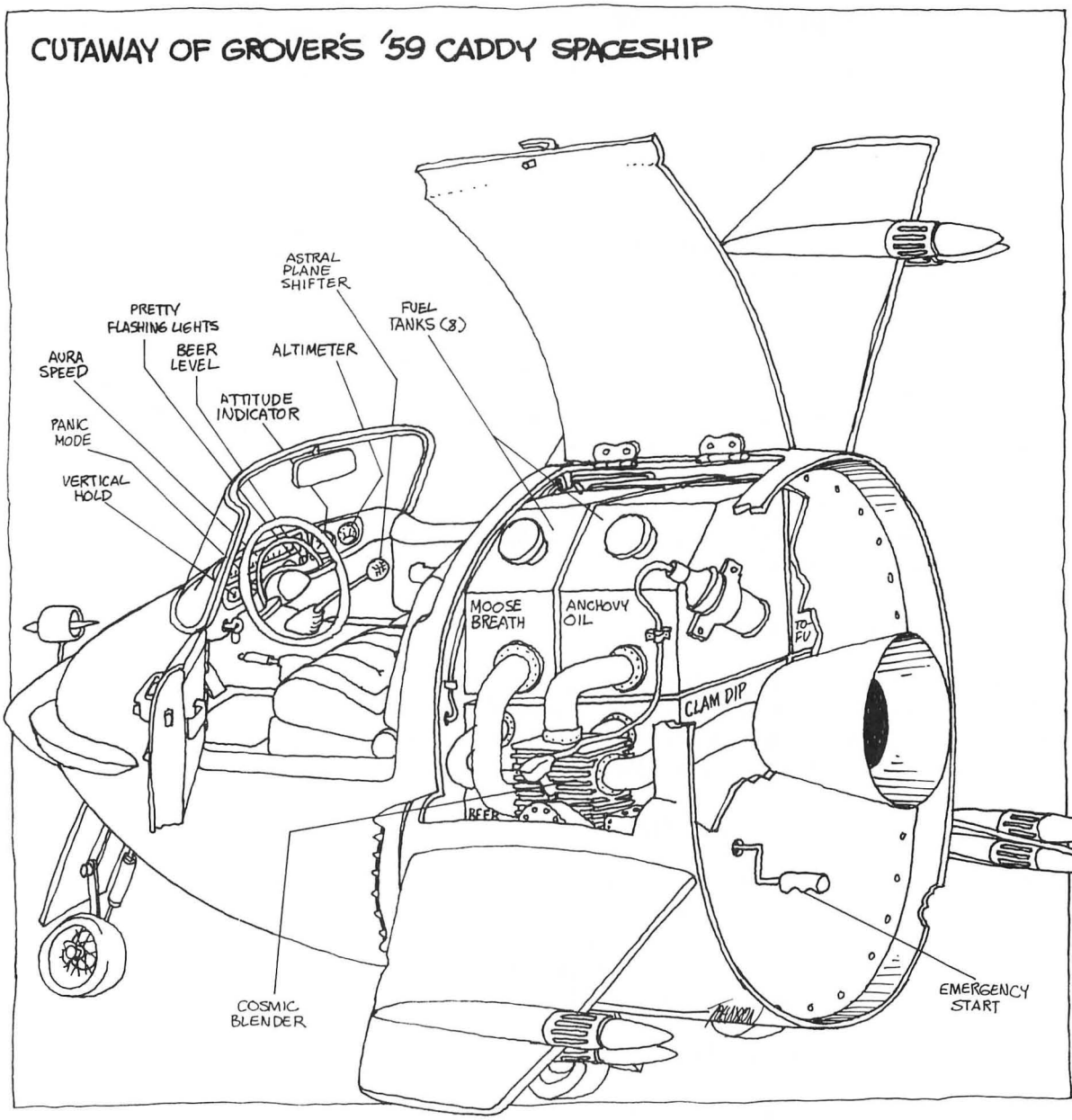
```
10 REM SOUND MACHINE
20 REM CERTAIN ARRANGEMENTS OF SOUND
   CAN BE MADE TO GENERATE INTERESTING
   AND EVEN EXCITING PATTERNS
```

```
 30 REM USING PURE TONES; WE DEFINE A
    SOUND AS ATTACKING/STABLE OR DECAYING
 40 REM VARIATIONS OF ATTACK AND DECAY PLUS
    A SMIDGEN OF STABLE SOUND MAKE
 50 REM GOOD GAME EFFECTS
 60 REM DEFINE ATTACK/STABLE AND DECAY
    BOTH IN TERMS OF PITCH AND VOLUME
 70 DIM U$(5)
 80 U$ = CHR$(156)
100 GRAPHICS 0:POKE 710,68:? CHR$(125);:POKE
    712,128:POKE 752,1:COLOR 32:PLOT 2,0:POKE
    709,15
110 POSITION 18,0:? "THE";:POSITION 15,1:
    ? "WONDERFUL";:POSITION 17,2:? "SOUND";:
    POSITION 17,3:? "SOUND";
120 POSITION 16,4:? "MACHINE";
130 POSITION 2,7:? "1) VOLUME ATTACK":
    ? "2) VOLUME DECAY":? "3) TONE DECAY":
    ? "4) TONE ATTACK"
140 ? "5) VOLUME AND TONE ATTACK":
    ? "6) VOLUME AND TONE DECAY":? "7) VOLUME
    ATTACK, TONE DECAY"
150 ? "8) VOLUME DECAY, TONE ATTACK":
    ? "9) STAR RAIDERS":? "10) SIREN":
    ? "11) WEIRD ZAP SOUND"
160 ? "12) POWER GENERATORS":
    ? "13) SOUND DABBLER MK.I"
165 ? "14) RETURN TO MENU"
170 TRAP 200:POSITION 0,22:? U$;U$;CHR$(127);
    CHR$(127);"OPTION";:INPUT A:IF A<1 OR A>14
    OR A<>INT(A) THEN 170
175 IF A = 14 THEN POKE 764,255:RUN "D:MENU"
180 ON A GOSUB 1000,2000,3000,4000,5000,6000,
    7000,8000,9000,10000,11000,12000,20000
190 GOTO 100
200 ? CHR$(253):GOTO 170
999 STOP
1000 REM VOLUME ATTACK
1010 FOR A = 0 TO 15
1020 SOUND 0,50,10,A
1030 FOR B = 0 TO 15:NEXT B
1040 NEXT A
1050 SOUND 0,0,0,0
1060 RETURN
2000 REM VOLUME DECAY
```

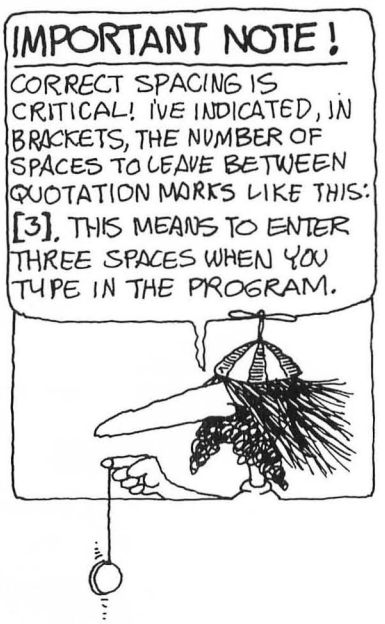
```
2010 FOR A=0 TO 15
2020 SOUND 0,50,10,15-A
2025 FOR B=0 TO 15:NEXT B
2030 NEXT A
2040 SOUND 0,0,0,0
2050 RETURN
3000 REM PITCH DECAY
3010 FOR A=1 TO 255 STEP 5
3020 SOUND 0,A,10,8
3025 FOR B=1 TO 15:NEXT B
3030 NEXT A
3040 SOUND 0,0,0,0
3050 RETURN
4000 REM PITCH ATTACK
4010 FOR A=1 TO 255 STEP 5
4020 SOUND 0,255-A,10,8
4025 FOR B=1 TO 15:NEXT B
4030 NEXT A
4040 SOUND 0,0,0,0
4050 RETURN
5000 REM ATTACK VOLUME AND PITCH
5010 FOR A=1 TO 50
5020 FOR B=1 TO 15
5030 SOUND 0,50-A,10,B
5040 NEXT B
5050 NEXT A
5060 SOUND 0,0,0,0
5070 RETURN
6000 REM DECAY VOLUME AND PITCH
6010 FOR A=1 TO 50
6020 FOR B=0 TO 15
6030 SOUND 0,A,10,15-B
6040 NEXT B
6050 NEXT A
6060 SOUND 0,0,0,0
6070 RETURN
7000 REM ATTACK VOLUME, DECAY PITCH
7010 FOR A=1 TO 50
7020 FOR B=0 TO 15
7030 SOUND 0,50+A,10,B
7040 NEXT B
7050 NEXT A
7055 SOUND 0,0,0,0
7060 RETURN
8000 REM DECAY VOLUME, ATTACK PITCH
```

```
 8010 FOR A = 1 TO 50
 8020 FOR B = 0 TO 15
 8030 SOUND 0,50 - A,10,15 - B
 8040 NEXT B:NEXT A
 8050 SOUND 0,0,0,0
 8060 RETURN
 9000 REM STAR RAIDERS
 9010 FOR A = 1 TO 10
 9030 SOUND 0,50,10,8
 9040 FOR B = 1 TO 50:NEXT B
 9050 SOUND 0,100,10,8
 9060 FOR B = 1 TO 50:NEXT B
 9070 NEXT A
 9080 SOUND 0,0,0,0
 9090 RETURN
10000 REM SIREN
10010 A = 1:B = 1
10015 FOR C = 1 TO 240
10020 B = B + A
10030 IF ABS(B)>15 THEN A = - A
10040 SOUND 0,45 + B,10,8
10050 NEXT C
10060 SOUND 0,0,0,0
10070 RETURN
11000 REM WEIRD ZAP SOUND
11010 FOR A = 1 TO 20
11020 FOR B = 1 TO 5
11030 FOR C = 1 TO 3
11040 SOUND 0,B*10 + C*2,10,B*C
11050 NEXT C
11060 NEXT B
11070 NEXT A
11080 SOUND 0,0,0,0
11090 RETURN
12000 REM S = TABLE DUAL PART = GENERATORS
12010 FOR A = 1 TO 500
12020 SOUND 0,70,12,8
12030 SOUND 1,71,12,8
12040 NEXT A
12050 SOUND 0,0,0,0
12060 SOUND 1,0,0,0
12070 RETURN
20000 REM DO YOUR OWN SOUND
20010 A1 = 0:A2 = 0
20020 B1 = 0:B2 = 0
```

```
20030 C1 = 0:C2 = 0
20040 GRAPHICS 0:POKE 752,1:COLOR 32:PLOT
      2,0:POKE 710,132:POKE 712,108:POKE 709,15
20050 POSITION 4,20:? "OPTION:ZERO VC 0
      SELECT:ZERO VC 1";:POSITION 4,20:
      ?"START = EXIT"
20060 POSITION 1,15:? "PATTERN: VOICE,PITCH,
      DISTORTION,VOLUME"
20070 POSITION 1,17:? "DIRECTION:N/A,UP/DOWN,
      LEFT/RIGHT,BUTTON"
20080 POSITION 10,3:? "JOYSTICK 0";:POSITION
      10,8:? "JOYSTICK 1";
20090 SOUND 0,A1,B1,C1:SOUND 1,A2,B2,C2
20100 POSITION 10,5:PRINT "VOICE 0: 0,";
      A1;",";B1;",";C1;" "
20110 POSITION 10,10:PRINT "VOICE 1: 1,";
      A2;",";B2;",";C2;" "
20120 A = STICK(0):B = STICK(1):C = STRIG(0):
      D = STRIG(1)
20130 A1 = A1 + (A = 13) - (A = 14):
      A2 = A2 + (B = 13) - (B = 14)
20140 B1 = B1 + 2*(A = 11) - 2*(A = 7):
      B2 = B2 + 2*(B = 11) - 2*(B = 7)
20150 C1 = C1 + (C = 0):C2 = C2 + (D = 0)
20160 IF A1>255 THEN A1 = 0
20170 IF A1<0 THEN A1 = 255
20180 IF A2>255 THEN A2 = 0
20190 IF A2<0 THEN A2 = 255
20200 IF B1>14 THEN B1 = 0
20210 IF B2>14 THEN B2 = 0
20220 IF B1<0 THEN B1 = 14
20240 IF B2<0 THEN B2 = 14
20250 IF C1>15 THEN C1 = 0
20260 IF C2>15 THEN C2 = 0
20270 A = PEEK(53279)
20280 IF A = 3 THEN A1 = 0:B1 = 0:C1 = 0
20290 IF A = 5 THEN A2 = 0:B2 = 0:C2 = 0
20300 IF A<>6 THEN 20090
20310 SOUND 0,0,0,0:SOUND 1,0,0,0
20320 RETURN
```


CUTAWAY OF GROVER'S '59 CADDY SPACESHIP
ASTRAL PLANE SHIFTER
PRETTY FLASHING LIGHTS
FUEL TANKS (8)
BEER LEVEL
ALTIMETER
AURA SPEED
ATTITUDE INDICATOR
PANIC MODE
VERTICAL HOLD
MOOSE BREATH
ANCHOVY OIL
TO FU
CLAM DIP
REFER
COSMIC BLENDER
EMERGENCY START

Appendix C: Myrtle the Turtle


IMPORTANT NOTE!
CORRECT SPACING IS CRITICAL! I'VE INDICATED, IN BRACKETS, THE NUMBER OF SPACES TO LEAVE BETWEEN QUOTATION MARKS LIKE THIS: [3]. THIS MEANS TO ENTER THREE SPACES WHEN YOU TYPE IN THE PROGRAM.

Here it is gang! Dr. Wacko's special deluxe super duper bonus game—Myrtle the Turtle!

I'll get into the workings of this original arcade game after you've had a chance to enjoy it.

Myrtle is a long program, the longest in this book. But because I've used some very special tricks (soon to be revealed) it will RUN on a 16K Atari.

So limber up your fingertips and start typing. I'll be back to watch you play in a few weeks. Don't forget to SAVE Myrtle to disk or cassette. I'm sure you'll want to play it again and again, and share the excitement with your friends and relatives.

If you've got a disk drive and the software version of Dr. Wacko's Miracle Guide, you're all set. Just press the right buttons and start enjoying Myrtle the Turtle!

What's it all about, Wacko?

Myrtle's just a turtle. All she wants to do is lay and fertilize eggs in the four "nests" located at each corner of the playfield—that's the *theme* of this game. Her *goal* is to create as many baby turtles as she can, and live to a ripe old age—level 20.

A Mean and Vicious Amoeba

But poor Myrtle is opposed (the *obstacle*) by a mean and viscious amoeba. This slimy, pulsing ameoba eats everything it lays its filia on, especially Myrtle and her eggs. And as the game progresses this slimy character becomes faster and smarter until, in level 10, it starts shooting reproductions of itself at Myrtle!

What's a Poor Turtle to Do?

Myrtle's got one trick up her webbed feet. Golden Time Holes appear in a pattern at different parts of the playfield. She can reach a Time Hole, step into it, teleport to one of her nests, and instantly escape from the hungry amoeba.

Movement

Control Myrtle with a joystick plugged into port 1.
Push the joystick up to move Myrtle up; down, to move down; left, to move left; right, to move right. Diagonal movement is also possible.

Getting Started

Now that you know how to control Myrtle, press the START button to begin play.

Teleportation

You've first got to reach a passing Time Hole to teleport to one of the four nests. The Time Hole moves about the playfield in a fixed pattern—it's up to you to figure this pattern out.

Once Myrtle is in the Time Hole, she's teleported to her nest by pointing the joystick toward the nest you've chosen. Only use the *four diagonal* joystick positions to teleport to a nest. This may take a little practice, but Slow POKE figured it out, so it's not that difficult!

Laying Eggs

Once Myrtle is safely at a nest, she'll lay an egg when you push the red trigger.

Fertilizing an Egg

Eggs can only be fertilized after Myrtle has layed eggs in all *four* nests. Just position Myrtle above an egg and press the trigger to fertilize it.

Moving On to Higher Levels of Play

To move on to the next level, Myrtle must fertilize at least one egg before the timer reaches zero.

When the eggs hatch, baby Myrtle's scramble from their nests into a community nest beneath the score and time readouts. The community nest can hold only nine baby turtles. If Myrtle creates more than nine babies you are awarded extra points for each additional baby hatched.

Replay

Press START to replay.

Scoring

- 1 Point for each teleportation
- 25 points for each egg layed
- 25 points for each egg hatched

Try to reach level 20. Happy fertilization!

Important programming note: Enter all underlined words and characters between quotation marks as inverse characters. To do this, press the Atari symbol key before you type the characters. An example of inverse characters is found in line 60: loadin'.

Myrtle the Turtle

```
 10 CLR :GOTO 40
 20 POSITION 4,7:? #6;VP;" ";:IF VP>HS THEN
    HS = VP
 30 POSITION 4,4:? #6;HS;" ";:RETURN
 40 DIM X(15),Y(15),S(7),C(2,16),EX(4),EY(4),CX(5),
    CY(5),E(4),L(15),MX(2)
 50 GRAPHICS 2:POKE 752,1:POKE 712,148:
    POSITION 7,3:PRINT #6;" myRtle":
    POSITION 9,4:PRINT #6;"is"
 60 POSITION 7,5:PRINT #6;"loadin":PRINT "[3]By
    Robert Kurcina & David Heller"
 70 PRINT "[3]Copyright 1983 Addison-Wesley Publishing"
 80 N = 0:N1 = 1:C = 2:N8 = 8:NX = 10:GOSUB 1250
 90 HT = 53278:SP = 1536:G0 = 53248:G1 = 53249:
    G2 = 53250:G3 = 53251:R0 = 704:R1 = 705:
    R2 = 706:R3 = 707:R4 = 708:R5 = 709
100 R6 = 710:R7 = 711:R8 = 712:P0 = 512:P1 = 640:
    P2 = 768:P3 = 896:PC = 53260:PF = 53261:FP = 53263
110 L = C:SX = NX:SY = N8:CX = N:CY = N:
    MX = MX(N1):MY = 80:YM = MY:YS = SY:
    VP = N:TM = N:F = N:CP = N:LA = N1:
    CL = N:ZZ = N:SL = N:LV = N1:T = N:CW = N
120 XX = N:YY = N:YB = N:MT = N:WC = N:SB = 4:
    P4 = N:P5 = N:ML = N:FOR A = N1 TO
    4:E(A) = N:NEXT A
```

```
130 POKE 756,ST/256:POKE 559,34 + N8:POKE
    R4,15:POKE R5,38:POKE R6,194:POKE
    R7,196:POKE R8,132
140 POKE R0,78:POKE R1,127:POKE R2,127:POKE
    R3,255
150 COLOR 154:PLOT N,N:DRAWTO 19,N:DRAWTO
    19,11:DRAWTO N,11:DRAWTO N,N
160 COLOR 186:FOR A = N1 TO C:PLOT A,A:DRAWTO
    19 - A,A:DRAWTO 19 - A,11 - A:DRAWTO
    A,11 - A:DRAWTO A,A:NEXT A
170 COLOR 154:PLOT 3,C:DRAWTO 16,C:DRAWTO
    16,9:DRAWTO 3,9:DRAWTO 3,C:PLOT
    9,3:DRAWTO 9,9:PLOT 11,3:DRAWTO 11,9
180 COLOR 186:PLOT NX,C:DRAWTO NX,9:COLOR
    28:PLOT N1,N1:PLOT 18,N1:PLOT N1,NX:PLOT
    18,NX
190 POSITION 4,3:? #6;" =>ABC";:POSITION 4,6:
    ? #6;"ABC";:POSITION 12,3:? #6;"DEF";:
    POSITION 12,6:? #6;"?@";
200 IF HS = N THEN 1170
210 MT = 112 - LV*C:FOR A = N1 TO 5:FOR B = N TO
    N1:FOR X = N1 TO NX:SOUND N,50 - A*X,NX,
    NX - X:NEXT X:POSITION 12,4:? #6;"[4]";
220 IF B = N1 THEN POSITION 12,4:? #6;MT;
230 NEXT B:NEXT A:IF L = N1 THEN 260
240 B = N:COLOR 43:FOR A = N1 TO L - 1:IF A>5
    THEN B = 3
250 PLOT A + 3 + B,N8:NEXT A
260 POSITION 4,4:? #6;"[5]";:POSITION 4,7:? #6;
    "[5]";:POSITION 12,4:? #6;"[5]";:POSITION 12,7:
    ? #6;"[5]";
270 GOSUB 20:POKE HT,N:POSITION 12,7:? #6;LV;:IF
    LV>5 THEN SB = N8
280 A = STICK(N):SX = SX + X(A):SY = SY + Y(A):IF
    SX<N OR SX>19 OR SY<N OR SY>11 THEN
    SX = SX - X(A):SY = SY - Y(A)
290 SOUND N1,N,N,N:TM = TM + N1:POKE
    R8,132 - C*(TM>MT/C):POSITION 12,4:?
    #6;MT - TM;"[1]";:IF TM = MT THEN POKE
    R8,N:GOTO 820
300 LOCATE SX,SY,ZZ:F = F + 1:IF F>C THEN F = N1
310 IF ZZ = N OR ZZ>31 AND ZZ<96 AND ZZ<>33
    AND ZZ<>72 THEN SX = SX - X(A):
    SY = SY - Y(A):SOUND N,SX*5 + SY*5 + NX,NX,6
```

```
320 POKE HT,N:D = USR(SP,PM + P0 + YS*N8 + 16,ST)
    :POKE G0,SX*N8 + 48:D = USR(SP,
    PM + P0 + SY*N8 + 16,ST + C(F,A)):YS = SY
330 IF A<>15 THEN POKE 77,N:SOUND N,40 + F*3,
    N8,NX - F*3:SOUND N,N,N,N:GOTO 410
340 IF STRIG(N) = N1 THEN 410
350 LOCATE SX,SY,ZZ:FOR X = N1 TO 10:SOUND
    N,120 - X*N8,NX,NX - X:NEXT X:IF ZZ = 33
    THEN 500
360 IF ZZ<>28 THEN 410
370 SOUND N,80,NX,15:FOR X = N1 TO 5:NEXT
    X:SOUND N,N,N,N:FOR X = N1 TO 4:IF SX = EX(X)
    AND SY = EY(X) THEN 390
380 NEXT X
390 E(X) = N1:POP :COLOR 33:PLOT SX,SY:FOR
    A = N1 TO 5:FOR B = N1 TO C:FOR X = N1 TO
    C:SOUND N,X*B*5,12,A*C:NEXT X
400 D = USR(SP,PM + P0 + SY*N8 + 16,
    ST + C(B,14)):NEXT B:NEXT A:SOUND N,N,N,N:
    VP = VP + 25:GOSUB 20:IF T = 5 THEN T = N
410 CP = PEEK(PC):IF CP = C OR CP = 6 THEN 820
420 IF CP = N8 OR CP = 12 THEN 770
430 IF CP = 4 THEN 550
440 IF NC = 0 THEN NC = N1:LA = LA + N1:IF LA>5
    THEN LA = N1
450 IF SX = CX(LA) AND SY = CY(LA) THEN
    LA = LA + N1:IF LA>5 THEN LA = N1
460 IF NC = N1 THEN CX = CX(LA)*N8 + 48:
    CY = CY(LA)*N8 + 16:D = USR(SP,
    PM + P2 + CY,ST + 120):POKE
    G2,CX:NC = C:SOUND N1,255,NX,15
470 IF NC = C THEN CL = CL + N1:D = USR(SP,
    PM + P2 + CY,ST + S(4 + F)):POKE R2,F*32 - N1
480 IF CL>NX THEN CL = N:NC = N:POKE
    G2,N:D = USR(SP,PM + P2 + CY,ST)
490 GOTO 600
500 FOR A = N1 TO 4:IF E(A) = N THEN POP :
    GOTO 410
510 NEXT A:LOCATE SX,SY,ZZ:IF ZZ<>33 THEN FOR
    X = N1 TO 4:SOUND N,200 - X*5,N8,X/C:NEXT
    X:SOUND N,N,N,N:GOTO 410
520 FOR X = N1 TO N8:SOUND
    N,200 - X*5,N8,X/C:NEXT X:SOUND
    N,N,N,N:COLOR 72:PLOT SX,SY
```

```
530 B = N:FOR X = N1 TO 4:LOCATE EX(X),EY(X),ZZ:IF
    ZZ<>72 THEN POP :GOTO 410
540 NEXT X:GOTO 820
550 FOR A = N1 TO 5:FOR X = N1 TO 3:FOR B = N1
    TO C:SOUND N,A*X*B*C + 40,NX,6:POKE
    R0,X*B*5:NEXT B:NEXT X:NEXT A:POKE R0,78
560 SOUND N,N,N,N:NC = N:CL = N:POKE
    G2,N:D = USR(SP,PM + P2 + CY,ST):POKE HT,N
570 VP = VP + N1:GOSUB 20:FOR A = N1 TO
    NX:SOUND N,NX - A,12,A + 5:NEXT
    A:D = USR(SP, PM + P0 + YS*N8 + 16,ST):
    SOUND N,N,N,N
580 A = L(STICK(N)):IF A<6 THEN
    SX = CX(A):SY = CY(A):YS = SY:POKE
    G0,SX*N8 + 48:D = USR(SP,
    PM + P0 + SY*N8 + 16,ST + 40):GOTO 440
590 A = A - 5:SX = EX(A):SY = EY(A):YS = SY:POKE
    G0,SX*N8 + 48:D = USR(SP,
    PM + P0 + SY*N8 + 16,ST + 40):GOTO 440
600 IF T = N THEN T = INT(RND(N)*4 + N1):IF E(T) = N
    THEN T = 5
610 B = INT(RND(N)*4 + N1):IF T = 5 AND E(B)>N AND
    RND(N)<0.1 THEN T = B
620 IF T<>5 AND ABS(SX*N8 + 48 - MX)<72 AND
    ABS(SY*N8 + 16 - MY)<72 THEN 1210
630 IF T<5 THEN X = SGN(EX(T)*N8 + 48 - MX)*SB:
    Y = SGN(EY(T)*N8 + 16 - MY)*SB
640 IF T = 5 THEN X = SGN(SX*N8 + 48 - MX)*SB:
    Y = SGN(SY*N8 + 16 - MY)*SB
650 MX = MX + X:LOCATE
    (MX - 48)/N8,(MY - 16)/N8,ZZ:
    IF ZZ = 33 OR ZZ = 72 THEN 790
660 IF ZZ>31 AND ZZ<96 THEN MX = MX - X
670 MY = MY + Y:LOCATE
    (MX - 48)/N8,(MY - 16)/N8,ZZ:
    IF ZZ = 33 OR ZZ = 72 THEN 790
680 IF ZZ>31 AND ZZ<96 THEN MY = MY - Y
690 POKE HT,0:D = USR(SP,PM + P1 + YM,ST):POKE
    G1,MX:D = USR(SP,PM + P1 + MY,
    ST + C(F,16)):YM = MY
700 IF PEEK(PF) = 4 THEN POKE HT,N:CL = N:
    NC = N:POKE G2,N:D = USR(SP,PM + P2 + CY,ST):
    SOUND N1,50,NX,15
710 IF SL = N THEN 280
```

```
720 ML = ML + N1:IF ML>5 + SL OR T = 5 THEN
    ML = N:XX = N:YY = N:P4 = N:P5 = N:POKE
    G3,N:D = USR(SP,PM + P3 + YB,ST):YB = N:
    SL = N:GOTO 280
730 IF SL = C AND F = C THEN
    P4 = SGN(SX*N8 + 48 - XX)
    *SB:P5 = SGN(SY*N8 + 16 - YY)*SB
740 POKE HT,N:D = USR(SP,PM + P3 + YB,ST):
    XX = XX + P4:YY = YY + P5:YB = YY:POKE
    G3,XX:D = USR(SP,PM + P3 + YY,ST + C(F,16))
750 SOUND N1,50 - F*C - SL*N8 - CL,NX,5:
    IF INT((XX - 48)/N8) = SX AND
    INT((YY - 16)/N8) = SY THEN 770
760 GOTO 280
770 POKE HT,N:POKE R8,15:SOUND N,50,12,15:FOR
    A = N1 TO 5:NEXT A:POKE R8,N:POKE G3,N
780 SOUND N,N,N,N:D = USR(SP,PM + P3 + YB,ST):
    XX = N:YY = N:YB = N:P4 = N:P5 = N:SOUND
    N1,N,N,N:GOTO 820
790 IF T = 5 THEN T = N:GOTO 690
800 E(T) = N:X = EX(T):Y = EY(T):T = N:FOR A = N1 TO
    5:FOR B = N1 TO C:SOUND
    N,B*NX + A*5 + NX,N8,5:
    NEXT B:NEXT A:COLOR 28
810 SOUND N,150,NX,15:FOR A = N1 TO 5:NEXT A:
    SOUND N,N,N,N:PLOT X,Y:GOTO 690
820 D = USR(SP,PM + P0 + YS*N8 + 16,ST):
    D = USR(SP,PM + P0 + SY*N8 + 16,ST + 40):
    FOR A = MX TO N STEP  - 4:POKE G1,A:
    SOUND N,A + NX,NX,NX
830 NEXT A:FOR A = CX TO N STEP  - N8:POKE
    G2,A:SOUND N,A + 30,NX,NX:NEXT A:
    D = USR(SP,PM + P3 + YB,ST):SL = PM + P2 + CY
840 D = USR(SP,PM + P1 + MY,ST):D = USR(SP,
    PM + P3 + YB,ST):D = USR(SP,SL,ST):FOR A = N1
    TO 3:FOR B = N1 TO 3:FOR X = N TO N1
850 POKE R1,A*B*NX + X*15:SOUND
    N,A*30 - B*5 + X,12,B + X*C + A*4:
    D = USR(SP,PM + P0 + SY*N8 + 16,
    ST + S(A + X)):NEXT X:NEXT B:NEXT A
860 D = USR(SP,PM + P0 + SY*N8 + 16,ST):
    POKE HT,N:CP = N:WC = N:TM = N:NC = N:T = N:
    CL = N:SL = N:CW = N:L = L - N1:ML = N
870 POKE R8,192:FOR A = N1 TO 4:LOCATE
    EX(A),EY(A),ZZ:E(A) = N:IF ZZ = 28 THEN 1080
```

```
880 IF ZZ = 72 THEN 900
890 FOR B = N1 TO 5:SOUND N,100 - B*20,NX,15:
    NEXT B:SOUND N,N,N,N:COLOR 28:PLOT
    EX(A),EY(A):GOTO 1080
900 FOR B = N1 TO 6:FOR X = N TO N1:FOR Y = N1
    TO 6:NEXT Y:SOUND N,B*NX - X*NX,
    NX,B*C + X:COLOR 72 + X*128:PLOT EX(A),EY(A)
910 NEXT X:NEXT B:COLOR 28:PLOT EX(A),EY(A):
    SX = EX(A):SY = EY(A):YS = SY:POKE
    G0,N:D = USR(SP,PM + P0 + SY*N8 + 16,ST + 40)
920 POKE R0,15:POKE G0,SX*N8 + 48:FOR B = N1 TO
    6:SOUND N,NX - B,NX,9 + B:NEXT B:SOUND
    N,N,N,N
930 FOR B = N1 TO 6:FOR X = N1 TO C:FOR Y = N1
    TO 5:NEXT Y:D = USR(SP,PM + P0 + SY*N8 + 16,
    ST + C(X,14)):NEXT X:NEXT B
940 X = SGN(NX - SX):Y = SGN(9 - SY):SX = SX + X:
    LOCATE SX,SY,ZZ:IF ZZ>31 AND ZZ<96 THEN
    SX = SX - X
950 SY = SY + Y:LOCATE SX,SY,ZZ:IF ZZ>31 AND
    ZZ<96 THEN SY = SY - Y
960 F = F + N1:IF F>C THEN F = N1
970 D = USR(SP,PM + P0 + YS*N8 + 16,ST):POKE
    G0,SX*N8 + 48:D = USR(SP,
    PM + P0 + SY*N8 + 16,ST + C(F,14)):YS = SY:
    SOUND N,40 + F*3,N8,NX - F*3
980 SOUND N,N,N,N:IF SX = NX AND SY = 9
    THEN 1000
990 GOTO 940
1000 CP = N1:FOR B = N1 TO 5:FOR X = N1 TO C:
    FOR Y = N1 TO 6:NEXT Y:D = USR(SP,
    PM + P0 + SY*N8 + 16,ST + C(X,14)):NEXT X:
    NEXT B
1010 VP = VP + 25:GOSUB 20:IF L + N1>9 THEN 1060
1020 L = L + N1:X = - N1:IF L>5 THEN X = N1
1030 D = USR(SP,PM + P0 + SY*N8 + 16,ST):
    SX = SX + X:SY = SY - N1:POKE G0,SX*N8 + 48:
    D = USR(SP,PM + P0 + SY*N8 + 16,
    ST + C(N1,14)):SOUND N,40,N8,NX
1040 FOR B = N1 TO 9:SOUND N,9 - B,NX,5:NEXT
    B:D = USR(SP,PM + P0 + SY*N8 + 16,ST):SOUND
    N,100,NX,15:X = L + 3:IF L>5 THEN X = L + 6
1050 COLOR 43:PLOT X,N8:SOUND N,N,N,N:POKE
    HT,N:GOTO 1080
```

```
1060 FOR B = N1 TO 3:FOR X = N TO N1:FOR Y = N1
     TO 6:NEXT Y:SOUND N,B*50 - X*30,12,15:
     D = USR(SP,PM + P0 + SY*N8 + 16,
     ST + S(B + X)):NEXT X
1070 NEXT B:D = USR(SP,PM + P0 + SY*N8 + 16,ST):
     SOUND N,N,N,N:POKE HT,N
1080 NEXT A:IF L = N THEN 1170
1090 IF CP = N1 THEN CP = N:LV = LV + N1:IF LV>20
     THEN 1140
1100 COLOR 32:PLOT 4,N8:DRAWTO N8,N8:PLOT
     12,N8:DRAWTO 15,N8:POKE R8,N:FOR A = N1 TO
     255:NEXT A:POKE R0,78
1110 SOUND N,100,NX,15:SX = NX:SY = N8:YS = SY:
     POKE G0,SX*N8 + 48:SOUND
     N,50,NX,15:D = USR(SP,PM + P0 + SY*N8 + 16,
     ST + C(N1,14))
1120 SOUND N,25,NX,15:FOR B = N1 TO 5:NEXT
     B:SOUND N,N,N,N:MX = MX((LV/C = INT
     (LV/C)) + N1):MY = 80:YM = MY:LA = N1:POKE
     R8,132
1130 GOTO 210
1140 POKE 20,N:FOR A = N1 TO
     NX:X = PEEK(R4):POKE R4,PEEK(R5):SOUND
     N,PEEK(20),NX,N8
1150 POKE R5,PEEK(R6):POKE R6,PEEK(R7):SOUND
     N,PEEK(20),NX,N8:POKE R7,PEEK(R8)
1160 SOUND N,PEEK(20),NX,N8:POKE R8,X:NEXT
     A:SOUND N,N,N,N
1170 FOR A = 250 TO N STEP  - 15:FOR X = N TO
     3:SOUND X,A + X,NX,15:NEXT X:NEXT A:FOR
     X = N TO 3:SOUND X,N,N,N:NEXT X
1180 COLOR 32:PLOT 4,N8:DRAWTO N8,N8:PLOT
     12,N8:DRAWTO 15,N8:GOSUB 20:IF HS = 0 THEN
     HS = - N1
1190 IF PEEK(53279)<>6 THEN POKE
     R4,PEEK(20):POKE R5,PEEK(20):GOTO 1190
1200 GOTO 110
1210 IF SL<>N OR LV<11 THEN 630
1220 XX = MX:YY = MY:YB = YY:
     P4 = SGN(SX*N8 + 48 - XX)*SB:
     P5 = SGN(SY*N8 + 16 - YY)*SB:SL = N1:IF LV>15
     THEN SL = C
1230 GOTO 630
1240 GOTO 1240
```

```
1250 MEMTOP = PEEK(741) + 256*PEEK(742) - N1
1260 PM = INT((MEMTOP - 1024)/1024)*1024:
     ADJTOP = PM + 384
1270 ST = PM + 1024:POKE 742,INT(ADJTOP/256):POKE
     741,ADJTOP - 256*PEEK(742)
1280 FOR X = N TO 512
1290 POKE ST + X,PEEK(57344 + X):NEXT X:POKE
     54279,PM/256:POKE 53277,C:FOR A = 512 TO
     1024:POKE PM + A,N:NEXT A
1300 RESTORE 1350:FOR A = 0 TO 127:READ X:POKE
     A + ST,X:NEXT A:FOR A = 208 TO 327:READ
     X:POKE A + ST,X:NEXT A
1310 FOR A = 1536 TO 1560:READ X:POKE A,X:NEXT
     A:FOR A = N1 TO 7:READ X:S(A) = X:NEXT A
1320 FOR A = N1 TO C:FOR B = N1 TO 15:READ
     X:C(A,B) = X:NEXT B:NEXT A:FOR A = N1 TO
     15:READ X:READ Y:X(A) = X:Y(A) = Y:NEXT A
1330 FOR A = N1 TO 4:READ X:READ
     Y:EX(A) = X:EY(A) = Y:NEXT A:FOR A = N1 TO
     5:READ X:READ Y:CX(A) = X:CY(A) = Y:NEXT A
1340 READ X:READ Y:C(N1,16) = X:C(C,16) = Y:FOR
     A = N1 TO 15:READ X:L(A) = X:NEXT
     A:MX(N1) = 184:MX(C) = 64:GRAPHICS 18:RETURN
1350 DATA 0,0,0,0,0,0,0,0,0,0,24,24,60,60,24,0,0,90,
     126,126,126,255,60,0,0,24,255,126,126,126,126,195
1360 DATA 33,63,62,126,126,62,63,33,0,24,255,126,
     126,255,60,0,132,252,124,126,126,124,252,132
1370 DATA 4,124,62,126,126,62,124,4,0,60,255,126,
     126,126,90,0,195,126,126,126,126,255,24,0
1380 DATA 0,0,16,56,56,0,0,0,0,16,124,56,124,0,0,0,
     32,62,124,126,126,124,62,32,36,60,239,98,70,
     247,60,36
1390 DATA 0,60,118,70,98,110,60,0,0,0,0,24,24,0,0,0,
     255,255,255,255,255,255,255,255,0,0,36,24,24,
     36,0,0
1400 DATA 0,126,126,126,126,126,126,0,0,85,85,117,
     85,85,0,0,0,212,20,92,84,212,0,0,0,0,69,69,69,
     69,114,0
1410 DATA 0,0,64,68,64,68,112,0,0,119,68,116,20,
     119,0,0,0,119,85,87,86,117,0,0,0,112,68,112,68,
     112,0,0
1420 DATA 0,117,37,37,37,37,0,0,0,23,180,247,84,23,
     0,0,0,0,64,0,64,0,0,0,0,66,24,60,60,24,66,0
1430 DATA 24,60,60,126,126,126,126,60
```

```
1440 DATA 104,104,133,204,104,133,203,104,133,207,
     104,133,206,160,0,177,206,145,203,200,192,8,2
     08,247,96
1450 DATA 40,68,80,120,216,312,0,40,40,40,40,96,16,
     96,40,64,56
1460 DATA 56,40,64,16,40,40,40,40,40,48,24,48,40,
     72,32,32
1470 DATA 40,72,24,40,0,0,0,0,0,0,0,0,1,1,1,-1,1,
     0,0,0,-1,1,-1,-1,-1,0,0,0,0,1,0,-1,0,0
1480 DATA 1,1,18,1,18,10,1,10,2,5,10,1,17,6,10,10,
     10,5,104,112,0,0,0,0,8,7,3,0,9,6,1,0,4,2,5
```

Wheew! That was a real whopper! But it was worth it. I know that you'll have hours of fun playing Myrtle.

Now, let's look a little closer at some of my demented programming.

Crunching the Program

I was able to squeeze this gigantic program into less than 16K of memory—with a little bit of help from trusting and well-oiled Clarence Compactor. This means that you can enjoy Myrtle on your Atari 400 or 600XL computer! Quite a feat, but not really difficult if you know the tricks of the trade.

You can use the compacting tricks I'm going to show you to squeeze really large programs into some pretty tight spots.

Compacting Trickeroos

- Remove all remark statements—REMs.
- Replace constants used more than three times with a variable. You'll save six bytes of memory each time you do this.

I've used this method to shrink Myrtle down to size. Take a look at line 80. See it? The first statement expression is N = 0. I've replaced the constant 0 with the variable N. If you look at the rest of the program you'll notice lots of Ns all over the place. These were all once 0s! I saved over 1000 bytes of memory by doing this.

- Be creative with your use of computer memory. Look at line 1270. I've placed the character set (ST) above the Player-Missile table. Not an orthodox thing to do, but it

worked, and made more room for the BASIC program.

- Concatenate lines into multiple statements. In other words, put more than one statement on each line, separated by colons. Line 110 is a great example of this concatenatious method. (What?)
- Set line numbers that your program branches to a lot through GOSUB and GOTO to predefined variables. For example, if your program goes to line 100 a bunch of times, make line 100 the variable L100 (L100 = 100) at the beginning of your program. Then replace all references to line 100 with L100. I didn't use this sneaky method in the Myrtle program, but if I did, a typical line would look like this: 1500 IF X = 5 THEN GOTO L100
- Keep your variable names as short as possible. The longer they are, the more memory they use.
- If you use the same word or words throughout your program such as START, replace it with a string variable. First you've got to dimension the string, then use the string variable to replace the word everywhere it appears in your program. Here's an example of this method:

```
10 DIM A$(5)
20 A$ = "START"
30 PRINT A$
```

Analyzing Myrtle

Get out your scalpel and surgical gloves and really dig into this program. It uses all the elements that you've learned in this book. It's challenging and educational to work through a program as complex as Myrtle. And you'll get a great sense of accomplishment once you've figured it all out. You'll also be weird and strange—like me!

You don't have to start your quest empty-handed, I've provided you with one helpful tool. A short routine that shows you all the characters used in this arcade game. Here it is:

```
0 GRAPHICS 2:POKE 756,ST/256:FOR X = 110 TO
  210:PRINT #6;CHR$(X);:NEXT X:STOP
```

To use this little gem first RUN Myrtle by typing: GOTO 10 <RETURN>. After Myrtle starts, press the BREAK key, then type GOTO 0 <RETURN> and you'll see all the weird shapes and characters used in this great game.

Appendix D: Smokey Peek's Pokes & Peeks

This appendix lists all the memory locations used in this book plus many that you'll find useful as you design the greatest BASIC aracade games in the universe.

The PEEK function lets "look" into a memory location and read its contents. The POKE statement lets you stuff information directly into a memory location.

16 & 53774

Used together *after each Graphics statement* disables BREAK key. Program example:

5 POKE 16,64:POKE 53774,64:GOTO 10

20

PEEKing this location will give you a source of ever changing numbers..Used in the *Flying Saucer* Player-Missile program, chapter 10.

88,89

Used in combination to find the first four screen pixel locations at the upper left corner of the screen. See POKEing Stuff to Graphics 19 in chapter 2.

540

Counts down to zero from a number you POKE into it. See the *Mystical Timing Routine* and *Sting* programs in chapter 9.

559

DMA Control Register (SDMCTL). Used to enable or disable direct memory access. POKEing with a value of 0 disables DMA. POKEing with a value of 1 turns DMA back on. Used to speed up computing process by as much a 30%. See *Building Block* #1 program, chapter 3. Also used in Player Missile Graphics. See *Dr. C. Wacko's Player-Missile Miracle Guide*, chapter 10.

580

Timer. POKEing 580 with 1 (POKE 580,1) at the beginning of your program purges the existing program from memory when the SYSTEM RESET key it pressed.

Color Registers

(See *Color Register POKES*, chapter 2.)

708

Controls Color Register 0

709

Controls Color Register 1

710

Controls Color Register 2

711

Controls Color Register 3

712

Controls Color Register 4

741,742

Free Memory High Address (MEMTOP) - See Dr. C. Wacko's Player-Missile Miracle Guide.

752

POKEing 752 with 0 gets rid of the cursor, POKE 752,1 puts cursor back on screen.

756

Character Address Base (CHBAS). Used as a pointer to redefined character set. See the *Building Block #1* program, chapter 3.

1536 to 1791

Free RAM. See the *Macine Language Flip-Flop Jogger* program in chapter 4.

53279

PEEK(53279) tells which special functon key has been pressed. A good example of its use can be found in the *Shoot-Out* program, chapter 9. This game begins when the player presses the START key, and a value of 6 causes the program to go from line 670 to line 675.

Values of PEEK(53279)

VALUE	KEY PRESSED
0	OPTION, SELECT or START
1	OPTION or SELECT
2	OPTION or START
3	OPTION
4	SELECT or START
5	SELECT
6	START

57344

Beginning location of Atari's character set. See *Building Block #1* program, chapter 3.

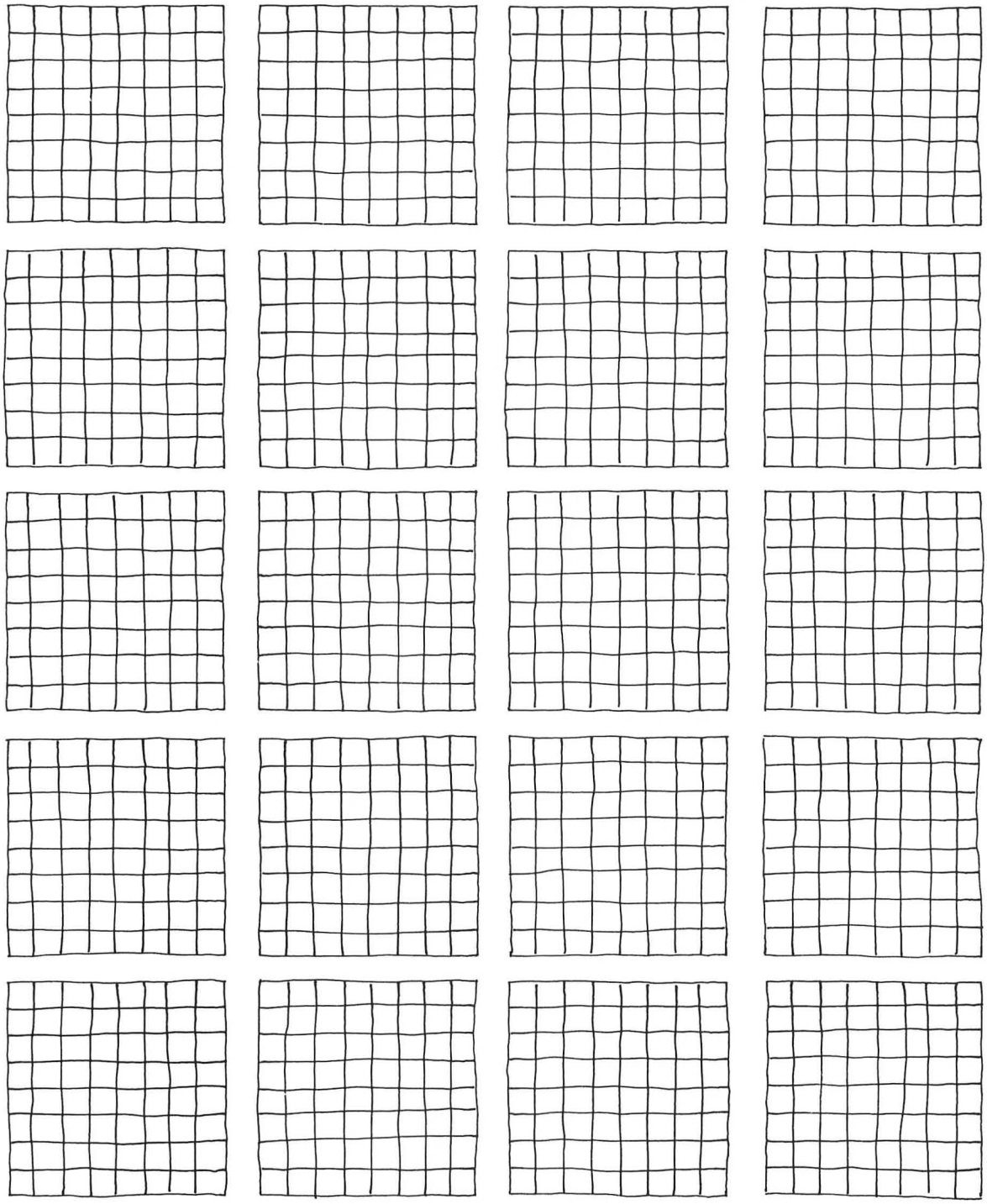
Player-Missile Locations

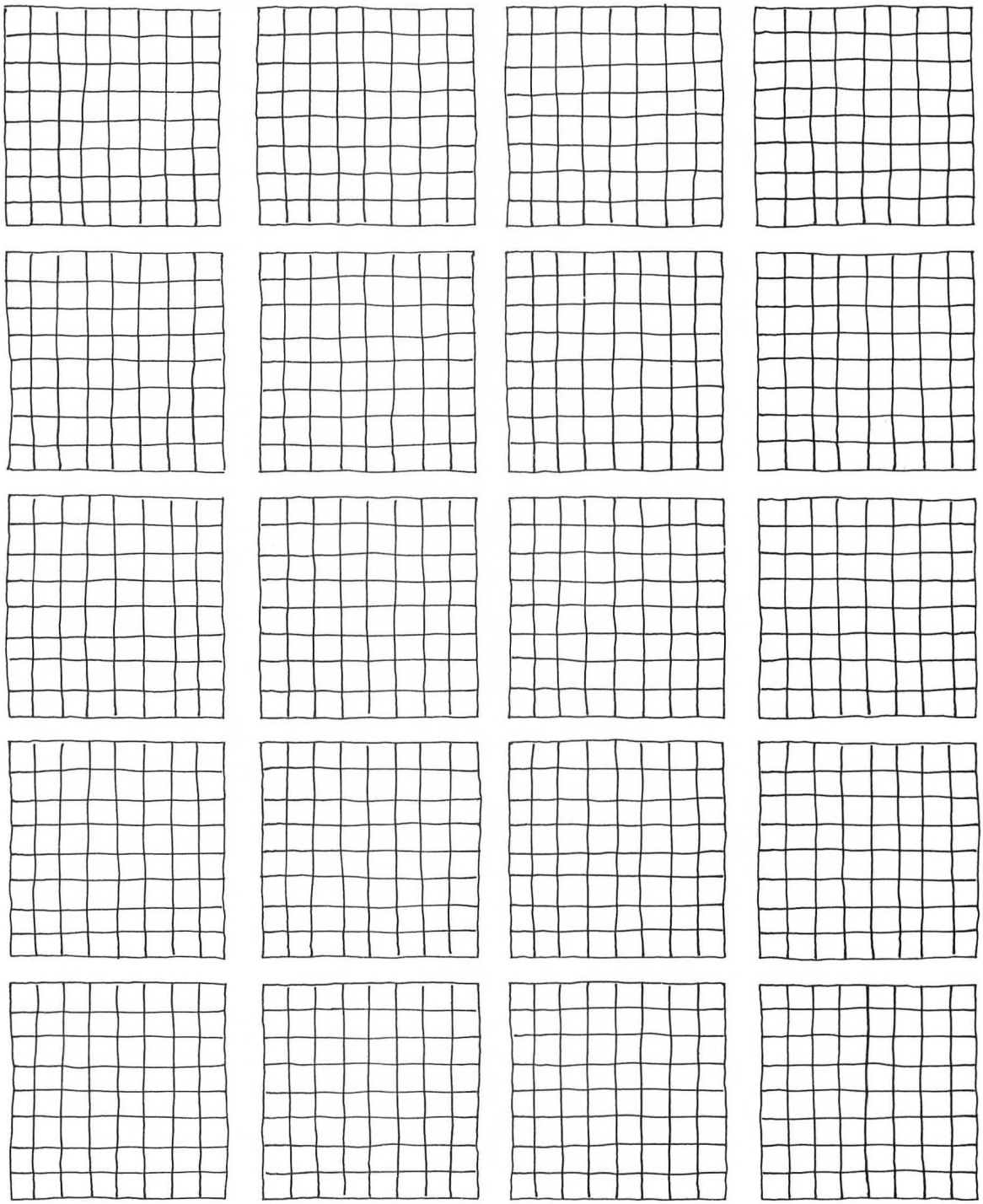
Player-Missile locations can be found in *Dr. C. Wacko's Player-Missile Graphics Miracle Guide*, chapter 10.

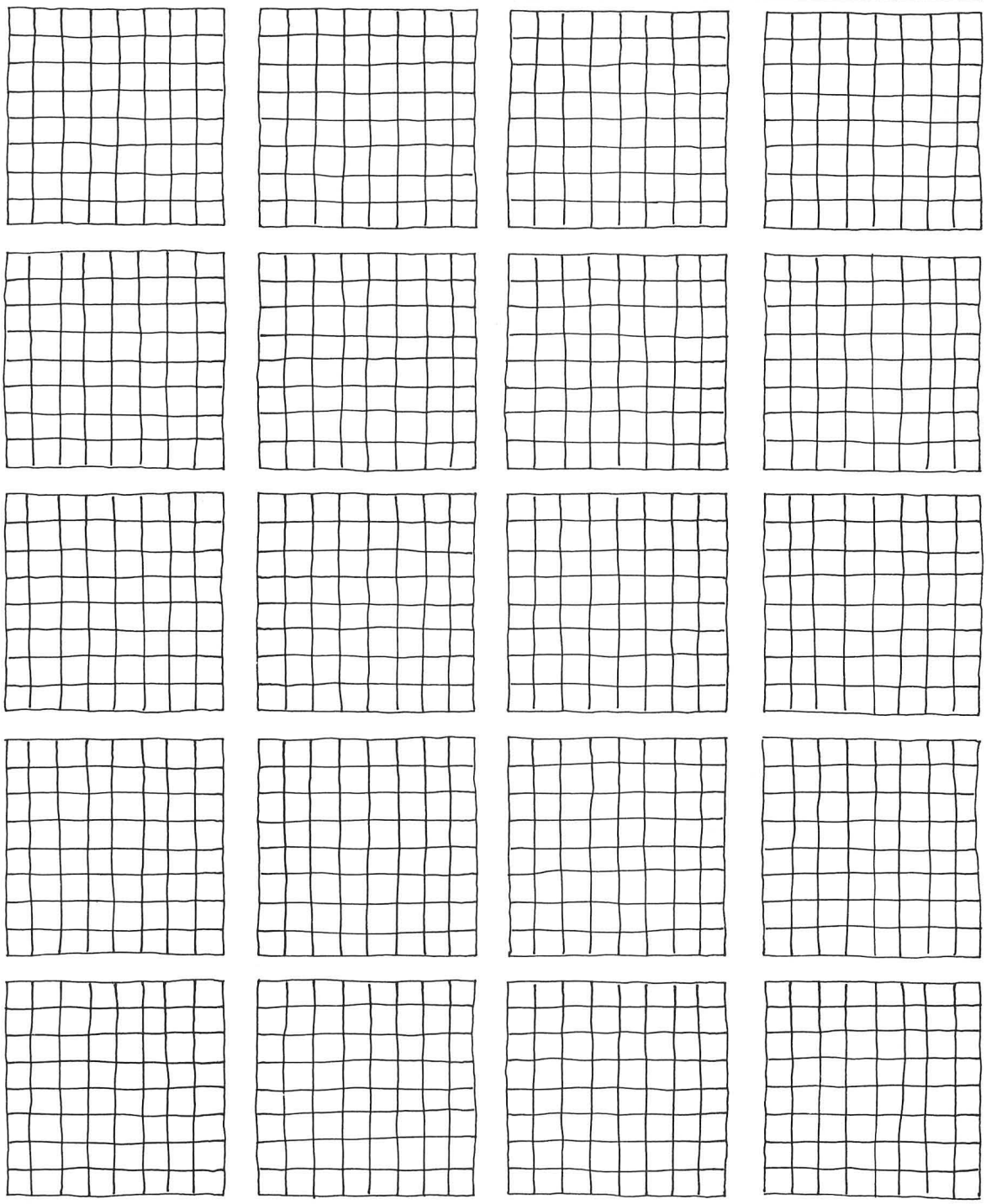
INDEX

Action 7, 10
ADJTOP 169-170
Adversaries 119-133
Amazing Feet Program (Weird Harold) 82-84
Animation
  18 Frames 117
  flip-flop 59, 71-89
  half character 80
  speed 72
  tester 71
  2 frames 80-81
ANTIC processor 56, 58, 161-162, 165, 166, 171, 175
ATASCII 32, 37
  codes, characters & keystrokes chart 189
Attack 140
Basic Sound Program 136
Big Frame-up Program 111-114
Binary Numbers, converting to decimal 47-48
Bong Program 42-44
Bouillabaise (Boolean) Logic 101-102, 106, 113, 124, 132, 181
Bounce 129-130
Boundary Bounceroo Program 130
Button, Red 65, 100-101, 104-105, 117
Character 10-11, 13, 163
Character Graphics 46-69
Character Set 46-53
Chaser 119-120
Chaser Program 122-128
Chords 146-148
COLOR & PLOT 24-29
Colorful Letters Program 33
Colorful Playfield Program 60-62
Coordinates 23, 91
Cursor 65, 91, 93-98, 163
  direction 96, 102-104
  movement speed 96
DATA Entry 72, 88
Decay 139-140
Difficulty Levels 13
Distortion 135
Drawing Behind Program 97-98
DRAWTO 24, 27-28
Dummy 77
Erasing Behind Program 94-96
Flip-Flop Jogger Program 73-74
  Machine Language Program 78-80
  String Program 86-87
  Grade A Program 87-88
Flying Saucer Program 176-182
FNTFILES 68
FONTLIST 66-67
FOR/NEXT Loop 29, 74, 139, 140
FR (Frame) 77, 83
Frame Set 109-110, 112-114
Free Memory High Address 168-169
Free RAM 75
Full Screen Display 22
Goals 7, 9, 10
GRAPHICS 20, 22, 29
Graphics Modes 20-25, 29-36
  demonstration program 20
IF/THEN 138-139
Joystick 65, 75, 99-107, 108-117
Know-it-all Program (Player-Missile Graphics) 164-165
LOCATE 36-37, 97-98, 129-130
Location 57344 48-50
Machine Language Machine 74-77, 84-86, 105
Mashed Monster Program 141-142
Mazecraze™ Program 18
MEMTOP 168-169, 171
Monster Maker Program 201-209
Movement 90-98
Music 142-152
Music Program ("The Sting") 149-152
Musical Chart 143-144
Musical Note Duration 148-149
Obstacles 7, 9, 10
OFFSET 50-52, 77
Opponents
  chaser/bouncer 119-120
  chasers 119, 121-126, 128
  low IQs 119
  roving robots 120, 131-133
  scaredy cats 119, 127-128
  vandals 119
OPTION = RVS 68
"Page 6" 75
Pitch 135, 145-146
Pixel Modes 21
Place Colors in Corners Program 40
Player-Missile Graphics 162-187
  memory allocation 166-174
  miracle guide 185-187
  10 commandments 174
Playfield 59-63
PMBASE 166, 167, 169, 170, 171
POKE 24, 26-27
POSITION 32
PRINT 21

PRINT#6	32-33
RAMTOP	166, 168
REM Memory Demonstration Program	49
Resolution	23, 166-167
Roboteroonie Program	131-133
Roving Robots	131-133
SC	38-41
Scaredy Cat Program	127-128
Score	14
SELECT = CLR	68
SGN	124-125
Shoot Out Program	154-160
Simple Movement Program	92-93
SL	77, 107
Sound	134-161
START = EXIT	68
Steinenwack	59
STRIG	100-102
Strings	84-88
Subroutine	140-141
Superbreakout™ Program	16-17
Superturnmeupsidedown Program	15
Text Modes	21, 29-36
Text Window	21-22
Theme	7-10
Time	12-13
Total Control Program	101-102
TRAP statement	136
Tron™ Program	18
"UNUSED" Memory	167, 168-170
USR Function	75, 76-77, 83, 106
Voice	135
Volume	135
Wackenstein	53, 55, 57, 59, 70, 71, 72, 115, 165, 173
Weird Harold	80-81, 115-116







WRITE YOUR OWN ACTION GAMES ON ANY ATARI® COMPUTER!

> [redacted] FPT
USA

Mastering Pac Man™ isn't much solace if you're a real computer gamer. What you *really* want to know is how to write your own Pac Mans. And now, for the first time ever, Dr. C. Wacko, world-renowned games programmer, tells all! From the necessary elements that shape any arcade game to the special programming tricks that can make your games sizzle, Dr. Wacko explains everything Atari® BASIC programmers need to create their own blockbusters.

What's more, the kindly Doctor gives you, absolutely free and without obligation, never-before-published programs that make designing and programming your games a snap! And some fiendishly difficult games designed by Wacko himself.

Enter the wacked-out world of Dr. Wacko and friends and you'll discover:

- Inside tips on designing arcade games that will turn humdrum ideas into bestsellers
- Essential programming techniques that mold your game into surefire winners
- Top-notch programs to start you on your way to arcade game fame and fortune.
- Much, much more

Caution: This book is fun and can be read by anyone who knows a little Atari® BASIC.

micro COMPUTER books

ADDISON-WESLEY PUBLISHING COMPANY

ISBN 0-201-11488-7